BODOR TIBOR

A COMMODORE 64 PROGRAMOZÁSÁNAK GYAKORLATA

KÖZVETLEN ELÉRÉSŰ LEMEZÁLLOMÁNYOK

3

SZÁMALK

A COMMODORE 64 PROGRAMOZÁSÁNAK GYAKORLATA III.

Közvetlen elérésű és relatív állományok

COMMODORE PROGRAMMING IN PRACTICE III.

Random and Relativ Files

TIBOR BODOR

COMMODORE PROGRAMMING IN PRACTICE

RANDOM AND RELATIVE FILES

Third volume

COMPUTING APPLICATIONS AND SERVICE CO. BUDAPEST, 1987

BODOR TIBOR

A COMMODORE 64 PROGRAMOZÁSÁNAK GYAKORLATA

KÖZVETLEN ELÉRÉSŰ ÉS RELATÍV ÁLLOMÁNYOK

Harmadik kötet

SZÁMÍTÁSTECHNIKA-ALKALMAZÁSI VÁLLALAT, BUDAPEST, 1987

COMMODORE 64 SOROZAT

Lektorálta: *Danis Miklós*
Supervised by *Miklós Danis*

ISBN 963 553 115 x (összkiadás)
ISBN 963 553 149 9 (III. kötet)

Tartalom

Contents

Előszó

A COMMODORE kétféle olyan lemezállományt ismer, amelyek elemei (rekordjai) közvetlenül elérhetők. Ezek egymástól a tárolási és a szervezési módban különböznek. Minthogy nincs általánosan elfogadott egységes elnevezésük, a továbbiakban átvesszük a COMMODORE szóhasználatát, és ezeket random, illetve relatív állományoknak fogjuk nevezni. Már a korábbi kötetekben is utaltunk rájuk, de akkor részletesen nem tárgyaltuk őket. Sorozatunk mostani kötetének célja e hiány pótlása, azaz a random és a relatív állományok használatának alaposabb ismertetése.

Az állományokkal kapcsolatos lemezkezelő parancsokon kívül programtervezési, adatfeldolgozási kérdésekre is kitérünk. Fontosnak tartjuk, hogy az Olvasó elmélyedjen ezek tanulmányozásában is, mivel a közvetlen elérésre épülő feldolgozás technikailag igen egyszerűen – noha a COMMODORE esetében kissé körülményesen – valósítható meg. A legfőbb nehézséget azonban nem ez, hanem a szervezési, tervezési kérdések jelentik.

Az állományok használatát, szokásunkhoz híven, működő mintaprogramokkal mutatjuk be. Feltételezzük, hogy az Olvasó abban a tekintetben nem kezdő, hogy a sorozat korábbi köteteit ismeri, viszont számottevő mikrogépes adatfeldolgozási és állománykezelési tapasztalatai még nincsenek. Ennek megfelelően a bemutatott példák nem valódi adatfeldolgozó programok, hanem a leegyszerűsített változataik. A programok hasznosságát alárendeltük annak a didaktikai szempontnak, hogy a megtanulandó állománykezelési módokat jól és áttekinthetően illusztrálják.

Javasoljuk, hogy az Olvasó két üres lemezt készítsen elő: egyet a random, egy másikat pedig a relatív állományok, illetve az ezeket kezelő programok számára. Így az állománykezelési módok kényelmesen kipróbálhatók a különböző keveredésekből, téves törlésekből, felülírásokból származó hibák veszélye nélkül.

Míg a soros állományok használata nemigen jellemző a COMMODORE-felhasználói környezetre, a random, illetve a relatív állományok már jóval nagyobb szerephez jutnak – a kisebb szervezeti egységekben ezekre épülnek a különböző nyilvántartások.

Budapest, 1986. A szerző

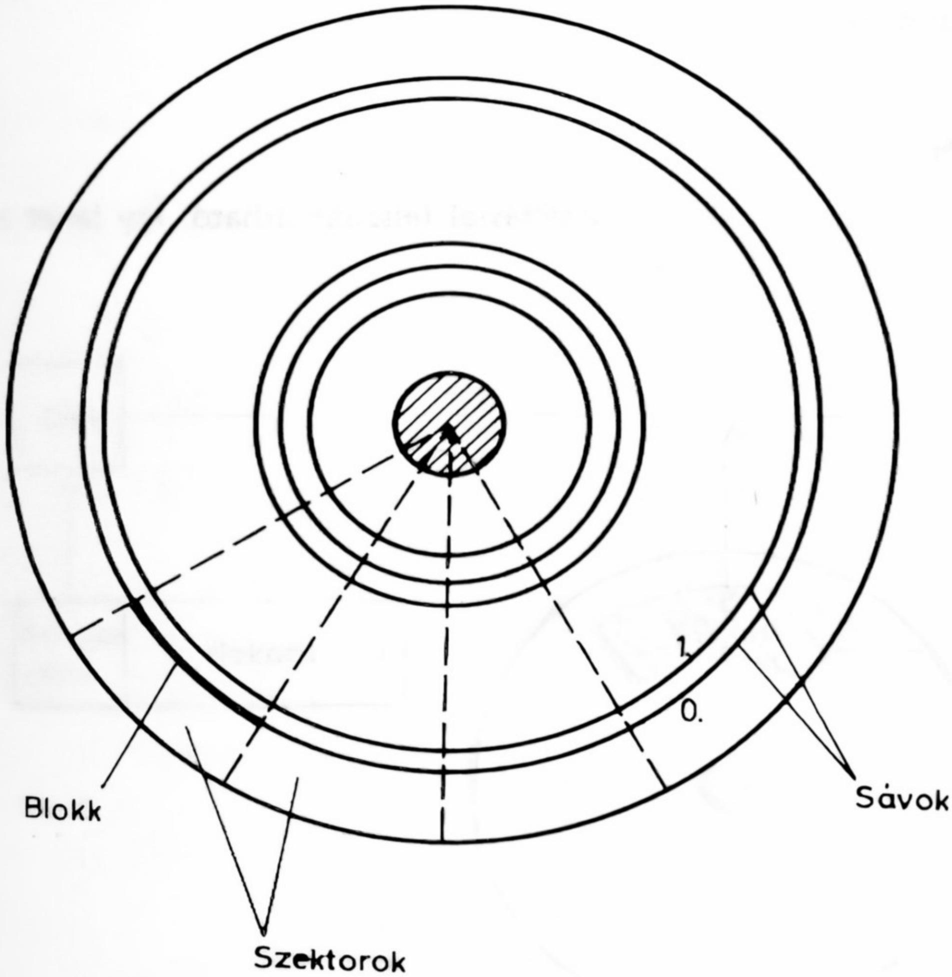
I. RÉSZ

RANDOM ÁLLOMÁNYOK

A random állományok kezelése

A random (azaz véletlenszerű) állományszervezési módot az teszi lehetővé, hogy a lemez minden egyes blokkja egyértelműen azonosítható egy sávcímmel és egy szektorcímmel, így az egyes blokkok tetszőleges sorrendben, közvetlenül elérhetők (**1.** ábra).

A COMMODORE lemezkezelő úgy valósítja meg a random állomány rekordjainak közvetlen elérését, hogy azokat egy-egy blokkon tárolja.



1. ábra. A lemez szervezése

Az ilyen állomány rekordjai a lemezen bárhol elhelyezkedhetnek, teljesen össze-vissza, látszólag véletlenszerűen. (Innen adódik a random elnevezés. Helytelen az elnevezést az elérési módból származtatni, ez ugyanis közvetlen, azaz direkt.)
E szervezési módnak a közvetlen elérés mellett egy másik előnye, hogy általa az állományok közötti hulladékblokkok is hasznosíthatók a lemezen.

A RANDOM ÁLLOMÁNY REKORDJAINAK ELÉRÉSE

A random állomány rekordjaihoz csak akkor férhetünk hozzá, ha ismerjük a tárolási helyüket, vagyis az egyes rekordokat tartalmazó blokkok címét, amint azt a 2. ábra mutatja.

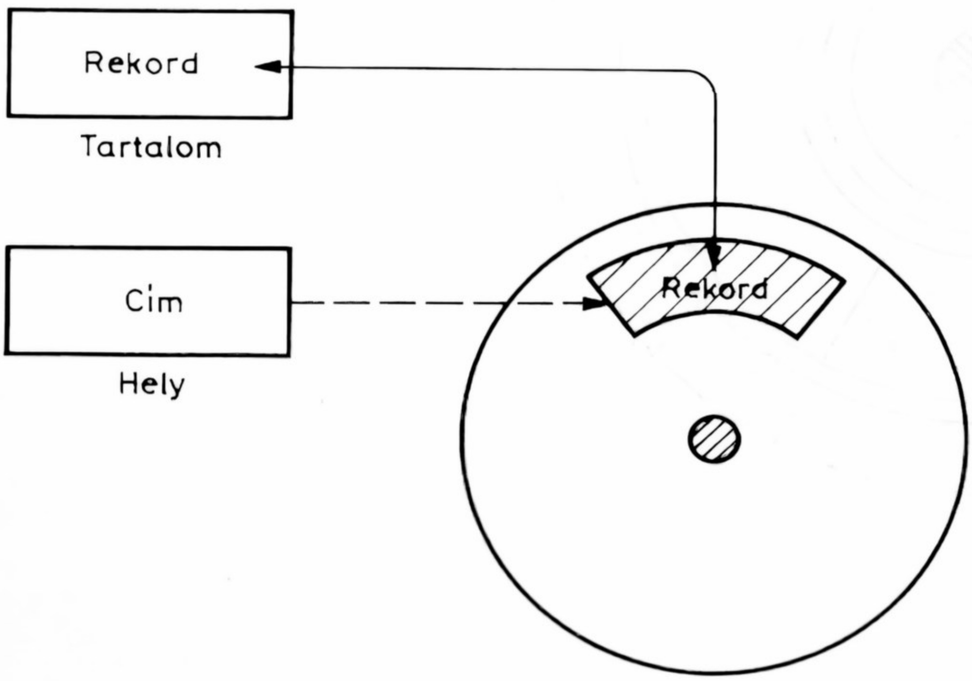
A rekordot úgy visszük fel a lemezre, hogy először keresünk neki egy helyet, azaz kiválasztunk egy blokkot, s ennek meghatározzuk a címét.

FIGYELEM!
Ügyelnünk kell arra, hogy a kiválasztott blokk szabad legyen, vagyis ne tartozzék valamilyen adatállományhoz, lemezen tárolt programhoz vagy a tartalomjegyzékhez, illetve a lemeztérképhez.

A kiválasztott blokkot le kell foglalnunk a random állomány számára, vagyis utasítanunk kell a lemezkezelőt, hogy egy ilyen értelmű bejegyzést tegyen a lemeztérképen. A lefoglalt blokkra már felvihetjük az adatrekordunkat.
Ha egy rekordot kívánunk beolvasni, először meg kell tudnunk, hogy melyik blokkon helyezkedik el, tehát meg kell tudnunk a rekordot tartalmazó blokk címét. Ennek ismeretében a rekord a blokkról leolvasható.

FIGYELEM!
Az olvasás sikertelen lesz, ha a blokk szabad volt.

A random állomány számára lefoglalt blokk egy utasítással felszabadítható. Így lehet a rekordot a lemezről törölni.

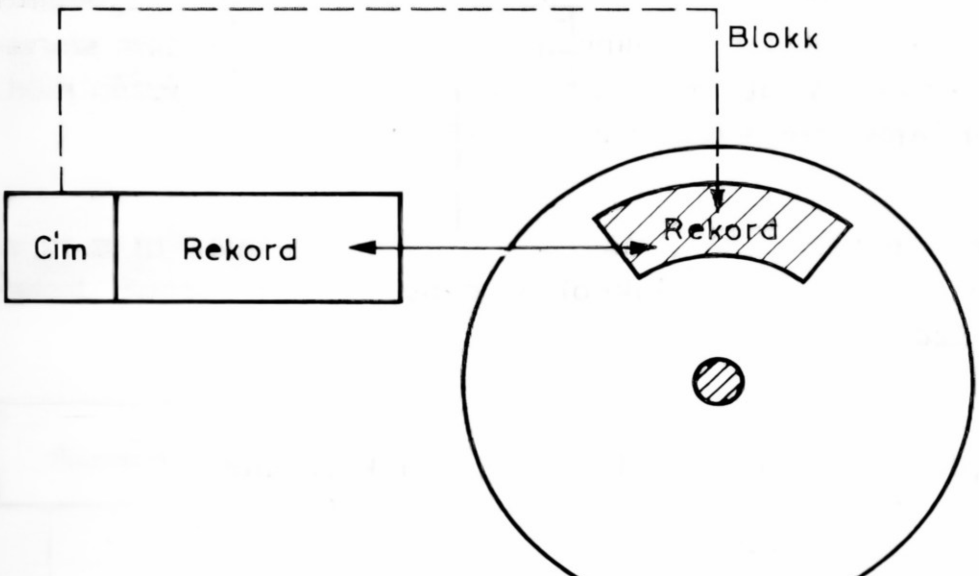


2. ábra. Rekord elérése

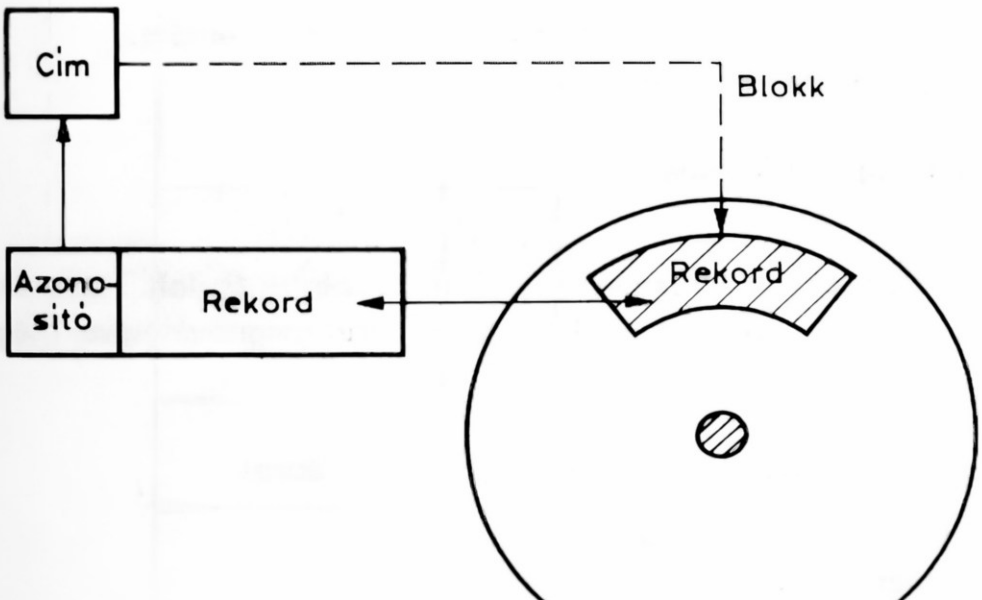
Megjegyezzük, hogy a lemezkezelő nem a rekordokat, hanem a blokkokat kezeli. A random állomány így az általa használt blokkot teljesen lefoglalja, akár kitöltik rekordjai a blokkot, akár nem. A lemezkezelő a felírást és az olvasást is pufferen keresztül hajtja végre.
A random állományok kezeléséhez elengedhetetlenül szükséges tehát, hogy a rekord és az őt tároló blokk között valamilyen kapcsolatot találjunk vagy létesítsünk.

A legegyszerűbb, ha ez a kapcsolat közvetlen (3. ábra), azaz ha a rekord azonosítója éppen a blokkcím. Ilyenkor például a 1215-ös azonosítójú rekord a lemez 12-es sávjának 15-ös szektorára kerül felvitelkor, és visszaolvasáskor ugyancsak az azonosítóból tudható meg, hogy a rekord hol keresendő. Ez a módszer adatszervezési okokból általában ritkán alkalmazható.

Gyakoribb, hogy a rekord és a blokk között közvetett kapcsolatot hozunk létre (4. ábra), azaz a blokkcímet a rekord azonosítójából valamilyen algoritmussal állítjuk elő.



3. ábra. Közvetlen kapcsolat



4. ábra. Közvetett kapcsolat

Ha például numerikus azonosítónk van, azt oszthatjuk 35-tel, illetve 16-tal, és az osztások maradéka képezi a sávcímet, illetve a szektorcímet. Ha a rekordazonosító 321-es, a blokkcím (6,1) lesz. Az ilyen leképezés azonban csak akkor kényelmes igazán, ha egyértelmű, és garantálva van, hogy a kiszámított blokk nem foglalt. Egy minden tekintetben megfelelő, a rekordazonosítóhoz alkalmazkodó leképező eljárást találni egyáltalán nem könnyű.

Ezért egy harmadik lehetőséget választunk, mely szerint a rekordazonosító és a blokkcím között semmilyen kapcsolat sincs, viszont külön nyilvántartást vezetünk arról, hogy az egyes rekordok a lemezen hová kerültek. Ezt a nyilvántartást indextáblának nevezzük. Az indextábla tulajdonképpen a logikai szerkezet és a fizikai elhelyezkedés között teremti meg a kapcsolatot.

Megjegyezzük, hogy a soros állományoknál sem feltétlenül folyamatosan, egymást követő blokkokban vannak az állomány rekordjai, vagyis az állomány lemezterületének nem kell szükségképpen összefüggőnek lennie. Itt azonban az állománykezelő rendszer megteremti a leképezést a rekordok között azzal, hogy összeláncolja a blokkokat. A random szervezés ebből a szempontból alacsonyabb szintű, mint a soros vagy a relatív szervezési mód, amelyeknél a leképezés megteremtése nem a programozó dolga.

A RANDOM ÁLLOMÁNY KEZELÉSE INDEXTÁBLÁVAL

Az indextábla tehát a rekordok azonosítóját, valamint az egyes rekordokat tároló blokkok sávcímét és szektorcímét tartalmazza.

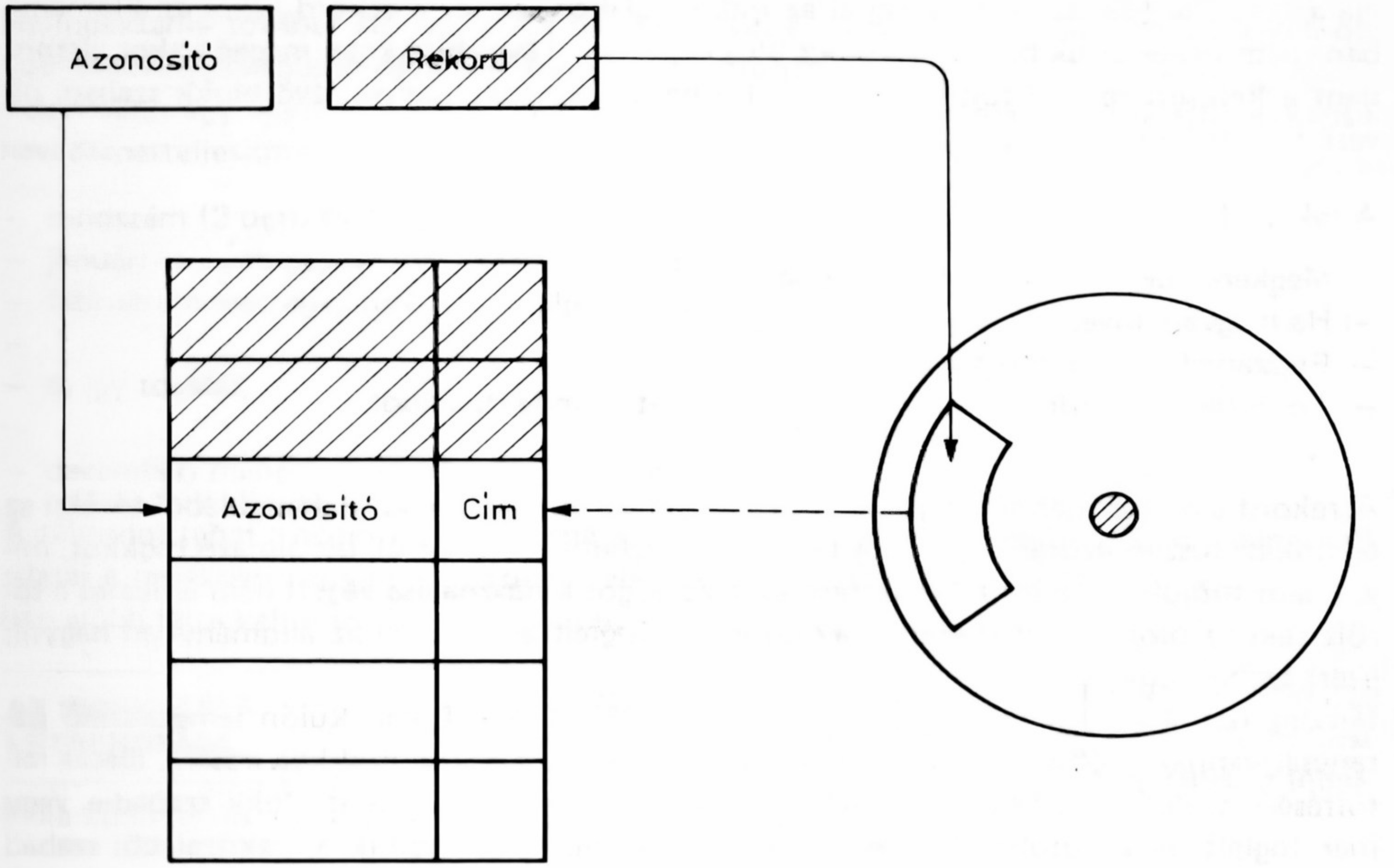
Egy rekord felviteléhez a műveletek egész sorát kell végrehajtanunk (5. ábra):

- Keresünk egy üres helyet az indextáblában.
- Ha találtunk egyet, oda bejegyezzük a rekord azonosítóját.
- Véletlenszerűen kiválasztunk egy blokkot a lemezen.
- Ha ez nem szabad, akkor megkeressük a legközelebbi szabad blokkot.
- Amikor már van szabad blokkunk, ezt lefoglaljuk a random állomány számára.
- A rekordazonosító mellé beírjuk az indextáblába a blokkcímet.
- Betöltjük a rekordot a pufferba.
- Felírjuk a puffer tartalmát a lefoglalt blokkba.

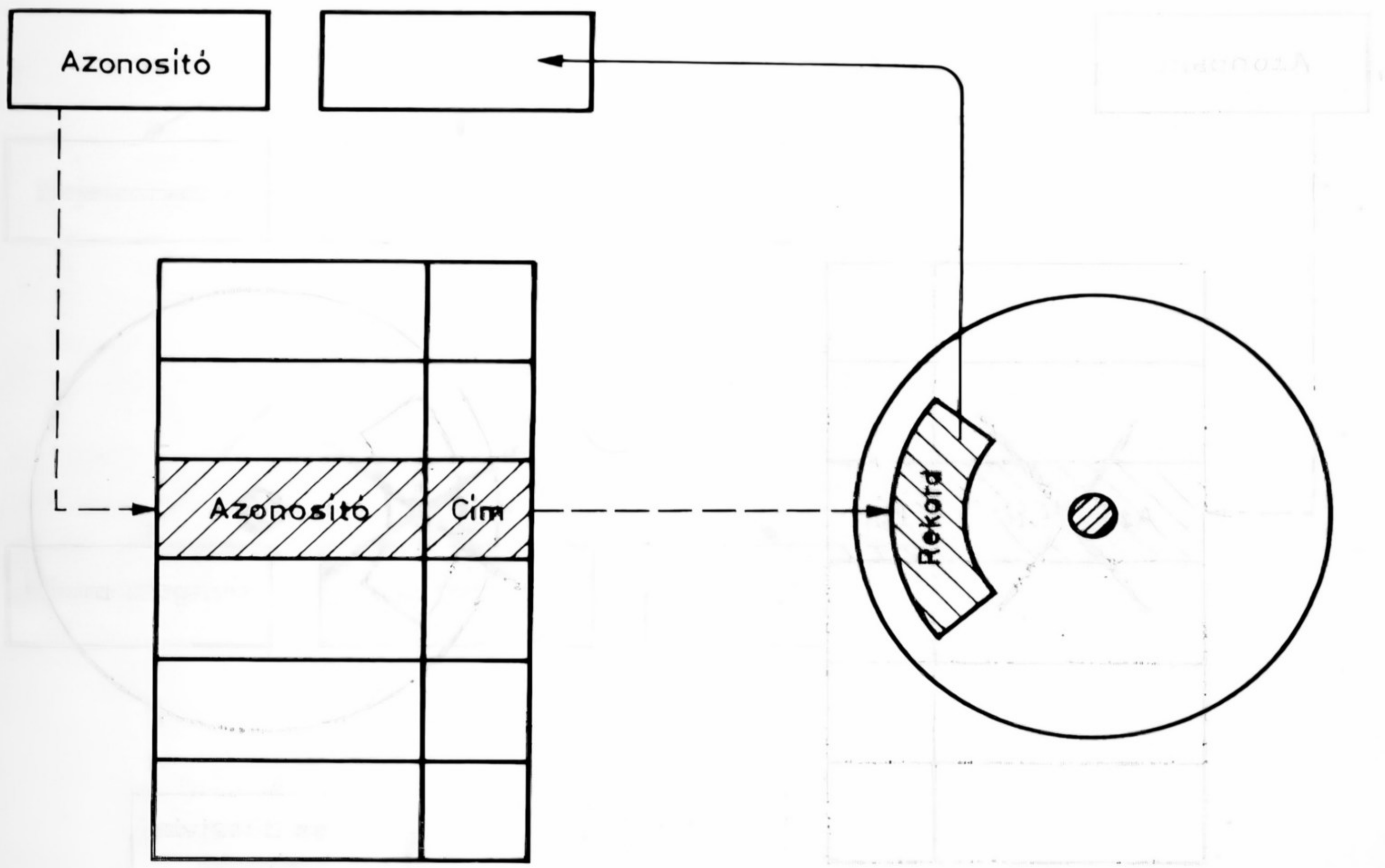
Természetesen nem tudjuk a rekordot felírni a lemezre, ha minden blokk foglalt. Hasonló a helyzet, ha az indextábla megtelt, bár ilyenkor esetleg a táblaméret megnövelésével még segíthetünk magunkon.

Egy rekord beolvasásakor a következő műveleteket kell elvégezni (6. ábra):

- Megkeressük a rekord azonosítóját az indextáblában.
- Ha megtaláltuk, kivesszük onnan a rekord blokkcímét.
- Az adott című blokkot betöltjük a pufferba.
- A rekordot kiolvassuk a pufferből.



5. ábra. Rekord felvitele



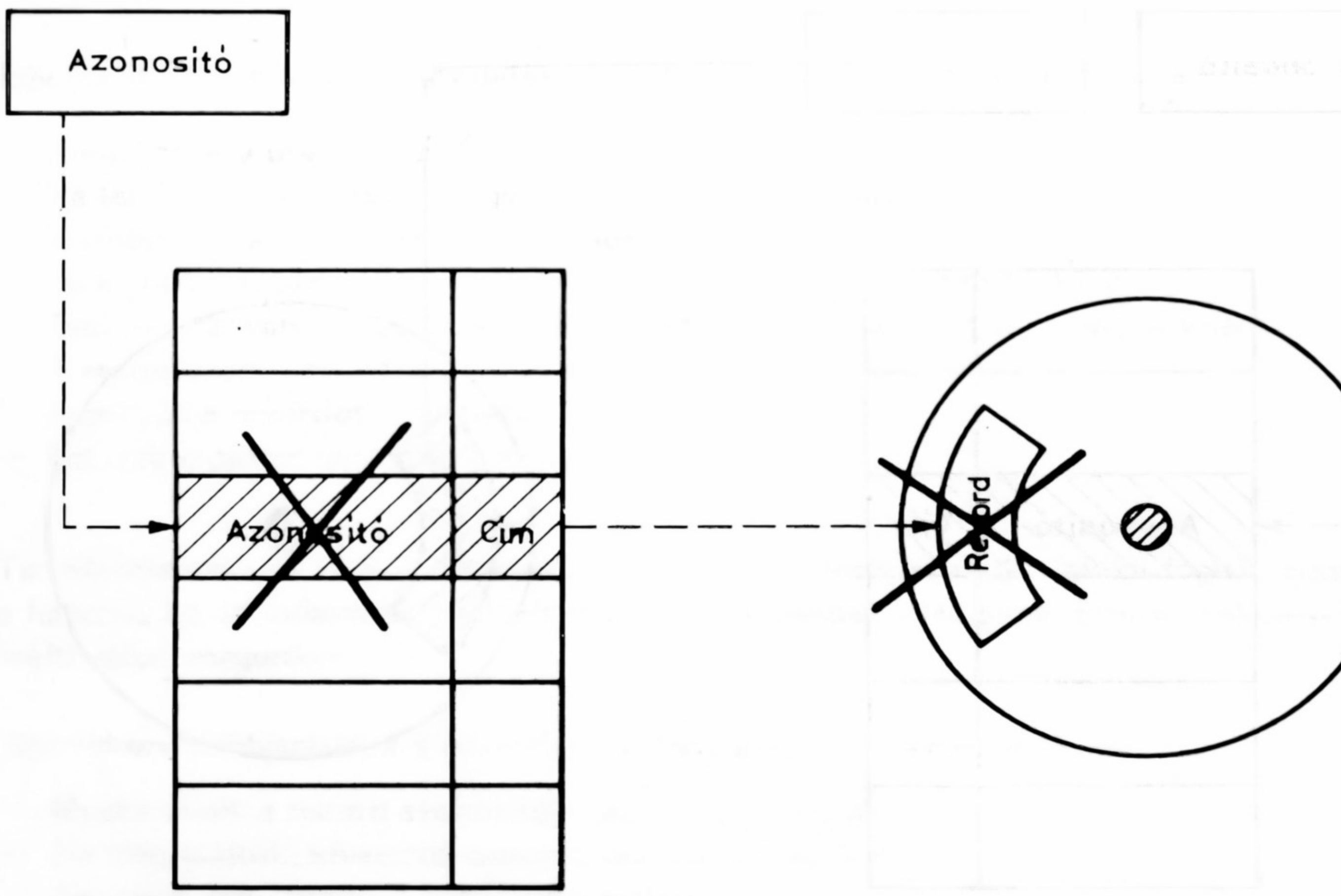
6. ábra. Rekord beolvasása

Ha a rekordazonosító nem szerepel az indextáblában, akkor a rekord nincs az állományban, nem olvashatjuk be, hiszen nincs blokkcímünk. Téves blokkcím megadásakor viszont nem a keresett rekord töltődik be a pufferba. Ha az adott címen lévő blokk szabad, olvasási hibát kapunk.

A rekord törlése egyszerűbb (7. ábra):

- Megkeressük az azonosítóját az indextáblában.
- Ha megvan, kivesszük onnan a blokkcímet.
- Felszabadítjuk az adott blokkot.
- Töröljük a rekordra vonatkozó bejegyzéseket az indextáblából.

A rekord szempontjából tulajdonképpen elegendő lenne csak az indextáblából törölni az adatokat, hiszen ezután már soha többé nem érhetnénk el az őt tartalmazó blokkot, mivel nem tudjuk a címét. A lemezterület gazdaságos kihasználása végett nem érdemes a törölt rekord blokkját elérhetetlenül ugyan, de foglalt állapotban az állományban hagyni; ezért szabadítjuk fel.
Mindez rendkívül bonyolultnak tűnik, de a COMMODORE-nak külön lemezkezelő parancsai vannak a blokk lefoglalására, felszabadítására, a puffer blokkba írására, illetve feltöltésére a blokkból. Sőt a gép megadja, hogy egy lefoglalni kívánt blokk szabad-e, vagy már foglalt, és ez utóbbi esetben még azt is közli, hogy melyik a legközelebbi szabad blokk. A géptől tehát a blokkok kezeléséhez megfelelő támogatást kapunk. Az indextábla és a rekordok pedig az eddig megismert utasításokkal kezelhetők.



7. ábra. Rekord törlése

Az indextábla további előnye, hogy használatával a rekordok és az őket tartalmazó blokkok között kölcsönösen egyértelmű kapcsolat létesíthető.

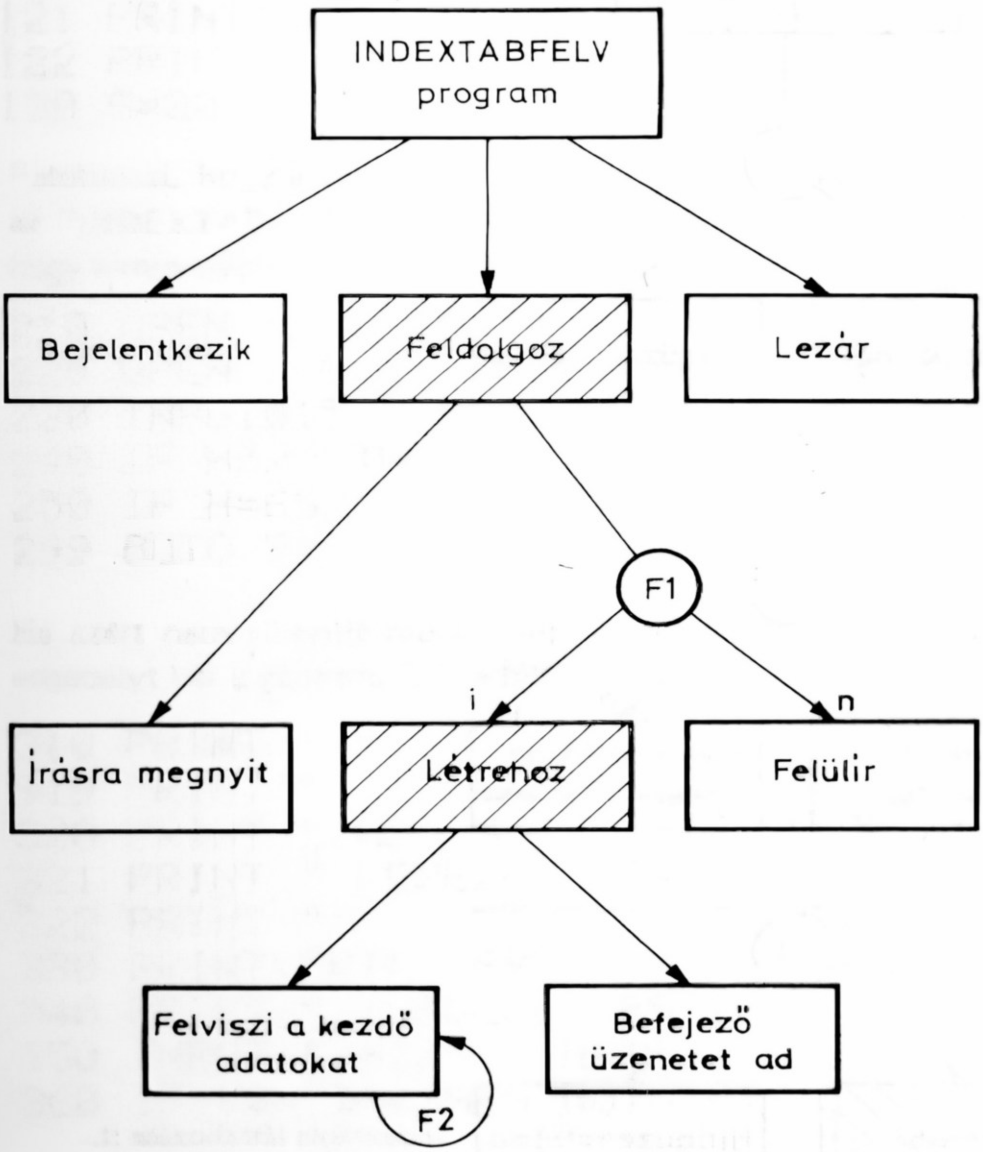
Tekintsünk egy egyszerű példát. Tároljuk egy random állományban a (vállalati) gépkocsik havi menetteljesítményét:

– rendszám (2 betű és 4 szám),
– januári menetteljesítmény (legfeljebb 9999 km),
– februári menetteljesítmény (legfeljebb 9999 km),
–
– és így tovább,
–
– decemberi menetteljesítmény (legfeljebb 9999 km).

A rekordot tehát a gépkocsi rendszáma azonosítja. Nyilvántartásunkban tehát a gépkocsik adatai a rendszám alapján közvetlenül elérhetők és karbantarthatók lesznek. Ehhez azonban előbb létre kell hoznunk az indextáblát.

AZ INDEXTÁBLA LÉTREHOZÁSA

A kérdés most az, hogy hol és milyen formában célszerű az indextáblát létrehozni. Ha nincsen sok rekordunk, a legalkalmasabb hely az indextábla számára a tár. Ott ugyanis az indextábla könnyen és gyorsan kezelhető.



8. ábra.
Indextábla létrehozása I.

Ha például a programunkat legfeljebb 30 gépkocsi nyilvántartására kell felkészíteni, a tárban egy-egy 30 elemű tömböt kell lefoglalnunk a rendszámok, a sávcímek és a szektorcímek tárolására.

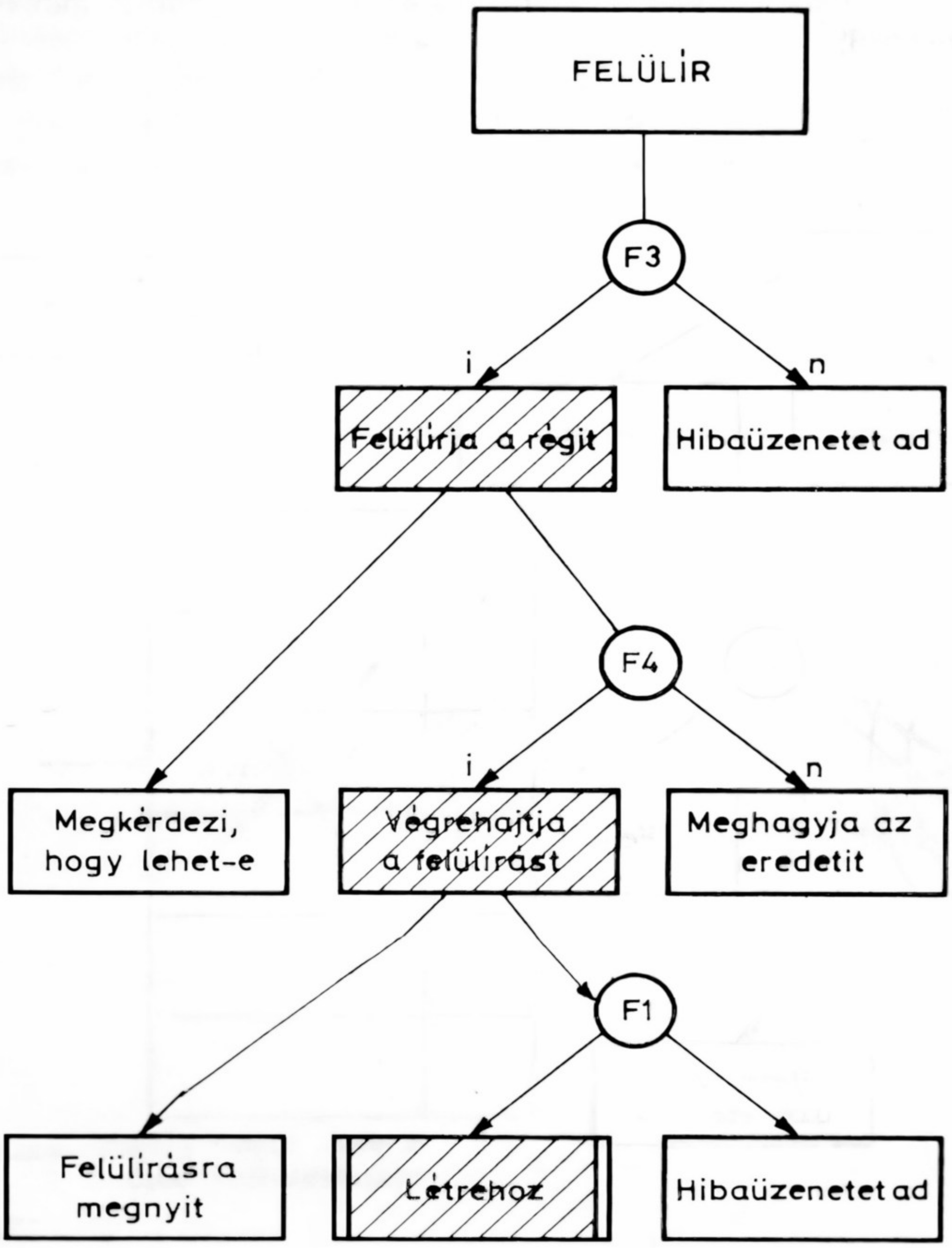
Természetesen szükségünk van arra, hogy az indextáblát lemezre kimentsük, hiszen máskor is használni akarjuk. Erre a soros adatállomány teljesen megfelelő. Valahányszor szükségünk van az indextáblára, betölthetjük a lemezről a tárba, és ott használhatjuk, amíg a programot le nem állítjuk. A feldolgozó programok ekkor úgy működnek, hogy a program elején beolvassák az indextáblát a tárba, szükség esetén módosítják, majd leállás előtt visszaírják a lemezre. Ha pedig törölni akarjuk a tábla tartalmát, felülírásra való megnyitással új indextáblát hozhatunk létre.

Nézzük meg azt az esetet, amikor egy eddig nem létező, új indextáblát kell létrehoznunk (8. ábra – 17. oldal).

A feltételek:

– F1: ha a megnyitás sikeres volt, azaz a H hibakód <20;
– F2: 1-től 30-ig egyesével nő (minthogy legfeljebb 30 gépkocsink lehet).

Ha az indextábla már létezik, akkor felülírással megsemmisíthetjük, és újat hozhatunk létre (9. ábra):



9. ábra.
Indextábla létrehozása II.
FELÜLÍR

Ahol:

- F3: ha a megnyitás azért nem sikerült, mert az állomány már létezik, azaz a H hibakód = 63;
- F4: ha az újbóli létrehozásra irányuló kérdésre a V$ válasz = "I".

Az újonnan létrehozott (üres) indextáblát olyan értékekkel kell feltölteni, amelyek nem téveszthetők össze az indextáblába később kerülő bejegyzések egyikével sem. (Ennek majd a kereséskor lesz szerepe.) Ezért a létrehozáskor a táblában az összes rendszám helyére csillagokat írunk, az összes sávcímnek és szektorcímnek pedig az 1-es értéket adjuk.

Megjegyezzük, hogy a létrehozáshoz nincs szükségünk magára az indextáblára, így nem is veszünk fel tömböket. Nézzük a programot!

Az indextáblát létrehozó programunk először bejelentkezik:

```
110 PRINT "▯"
111 PRINT
112 PRINT "  ▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬"
113 PRINT "  ▪ INDEXTABLA GENERALASA ■"
114 PRINT "  ▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬"
115 PRINT
120 PRINT
121 PRINT "       30 GEPKOCSIRA"
122 PRINT
130 S=30
```

Feltételezi, hogy a tábla még nem létezik, ezért írásra (létrehozásra) nyitja meg számára az "INDEXTABLA" lemezállományt. Figyeli azonban a parancscsatorna H hibakódját, hogy a megnyitás sikeres volt-e:

```
210 OPEN 15,8,15
220 OPEN 2,8,2,"INDEXTABLA,SEQ,WRITE"
230 INPUT#15,H,H$
240 IF H<20 THEN GOTO 610
250 IF H=63 THEN GOTO 310
299 GOTO 510
```

Ha azért nem sikerült megnyitnia az állományt, mert az már létezik, akkor a program engedélyt kér a gépkezelőtől a tábla törlésére és újbóli létrehozására:

```
310 PRINT
319 PRINT "       ▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬"
320 PRINT " AZ  ▪ INDEXTABLA ■  MAR A";
321 PRINT " LEMEZEN VAN!"
322 PRINT "       ▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬"
330 PRINT:PRINT:PRINT
340 PRINT " TOROLJEM ES KESZITSEK UJAT";
350 INPUT " =I/N=  N▮▮▮▮";V$
360 IF V$="I" THEN GOTO 410
```

Ha nemleges V$ választ kap, akkor nem hozza létre az indextáblát tartalmazó állományt, hanem befejezi a futást:

```
370 PRINT
371 PRINT " A LEMEZEN AZ EREDETI";
372 PRINT " INDEXTABLA MARAD."
373 PRINT
399 GOTO 910
```

Ha viszont az indextábla újbóli létrehozása mellett döntöttünk, akkor felülírásra nyitja meg az "INDEXTABLA" állományt. (Előbb persze lezárja az írásra sikertelenül megnyitott adatcsatornát.) A parancscsatornát a biztonság kedvéért most is figyeli:

```
410 CLOSE 2
420 OPEN 2,8,2,"@:INDEXTABLA,SEQ,WRITE"
430 INPUT#15,H,H$
440 IF H<20 THEN GOTO 610
```

Ha a megnyitás sikertelen volt, a képernyőre írja a parancscsatornáról kapott H hibakódot és H$ hibaüzenetet:

```
510 PRINT
520 PRINT " ■ SIKERTELEN MEGNYITAS ■"
530 PRINT : PRINT H;H$
599 GOTO 910
```

Ugyanezt teszi, ha az írásra való megnyitáskor a hibát nem az okozta, hogy az állomány már létezik a lemezen.

Sikeres megnyitás esetén — akár írásra, akár felülírásra vonatkozott — felír a lemezre 30 darab rekordot, mindegyikbe hat csillagból álló rendszámot, valamint a hozzá tartozó blokkcímet, amely egységesen az 1-es sáv 1-es szektorát határozza meg:

```
610 FOR I=1 TO S
620 PRINT#2,"******"
630 PRINT#2,1
640 PRINT#2,1
650 NEXT I
```

Megjegyezzük, hogy a karbantartás során majd igen nagy hasznát vesszük annak, hogy a táblában megadott kezdőcímek a lemez elején vannak. Azért nem az 1-es sáv 0-s szektorát adtuk meg, mert azt különleges célokra szokás fenntartani. (Például a lemez azonosítására; jelszók, hozzáférési módok tárolására; stb.)

Az állomány létrehozása vagy újbóli létrehozása után a program kijelentkezik:

```
810 PRINT
811 PRINT " ▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄"
812 PRINT " ■ + KESZ + ■"
813 PRINT " ▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀"
814 PRINT
```

Végül leáll:

```
910 CLOSE 2
920 CLOSE 15
999 END
```

Az indextáblát létrehozó programunkat gépeljük be, mentsük ki a lemezre például "INDEXTABFELV" néven, majd próbáljuk ki.

Ha lefuttatjuk, létrejön az "INDEXTABLA" soros állomány. Erről a lemeztartalom lekérdezése útján megbizonyosodhatunk.

A felülírást egyszerűen kipróbálhatjuk; indítsuk el a programot másodszor is. Ha most a program megkérdezi, hogy: "TOROLJEM ES KESZITSEK UJAT?", akkor adjunk igenlő választ.

Amikor harmadszor is elindítjuk a programot, kipróbálhatjuk, hogy mi történik, ha a fenti kérdésre tagadó választ adunk.

FIGYELEM!

A programban több helyen szerepelnek képernyőkezelő karakterek. Ilyenek például a 320-as sorban az inverz írást bekapcsoló (RVS ON), illetve kikapcsoló (RVS OFF); vagy a 350-es sorban a kurzort (CRSR) balra mozgató jelek. Ezek használatát a „SOROS LEMEZÁLLOMÁNYOK" c. kötetben már tárgyaltuk. Minthogy a következő programjainkban is igen gyakran előfordulnak, emlékeztetőül felsoroljuk begépelésük módját, a hatásukat és a nyomtatásban megjelenő formájukat:

```
"[HOME]"      = [HOME]              : KURZOR BAL FELSO SAROKBA
"[CLR]"       = [SHIFT]+[CLR]       : KEPERNYO TORLESE
"[RVS ON]"    = [CTRL]+[RVS ON]     : ATTERES INVERZ IRASRA
"[RVS OFF]"   = [CTRL]+[RVS OFF]    : VISSZATERES NORMAL IRASRA
"[CRSR DOWN]" = [CRSR]              : KURZOR LE
"[CRSR UP]"   = [SHIFT]+[CRSR]      : KURZOR FEL
"[CRSR RIGHT]"= [CRSR]              : KURZOR JOBBRA
"[CRSR LEFT]" = [SHIFT]+[CRSR]      : KURZOR BALRA
```

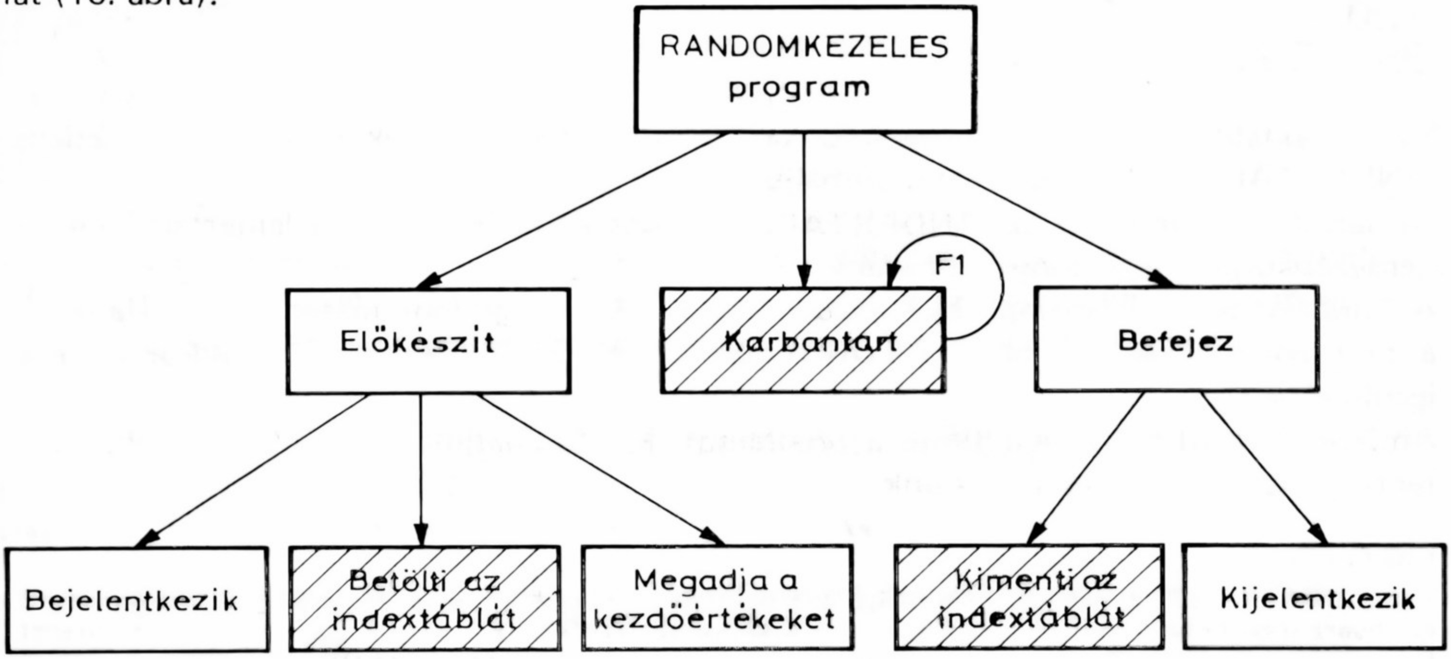
Van tehát a lemezen egy üres, azaz alapértelmezésünk szerinti adatokkal feltöltött indextáblánk. Hozzáfoghatunk a random állomány létrehozásához.

A RANDOM ÁLLOMÁNY KEZELÉSE

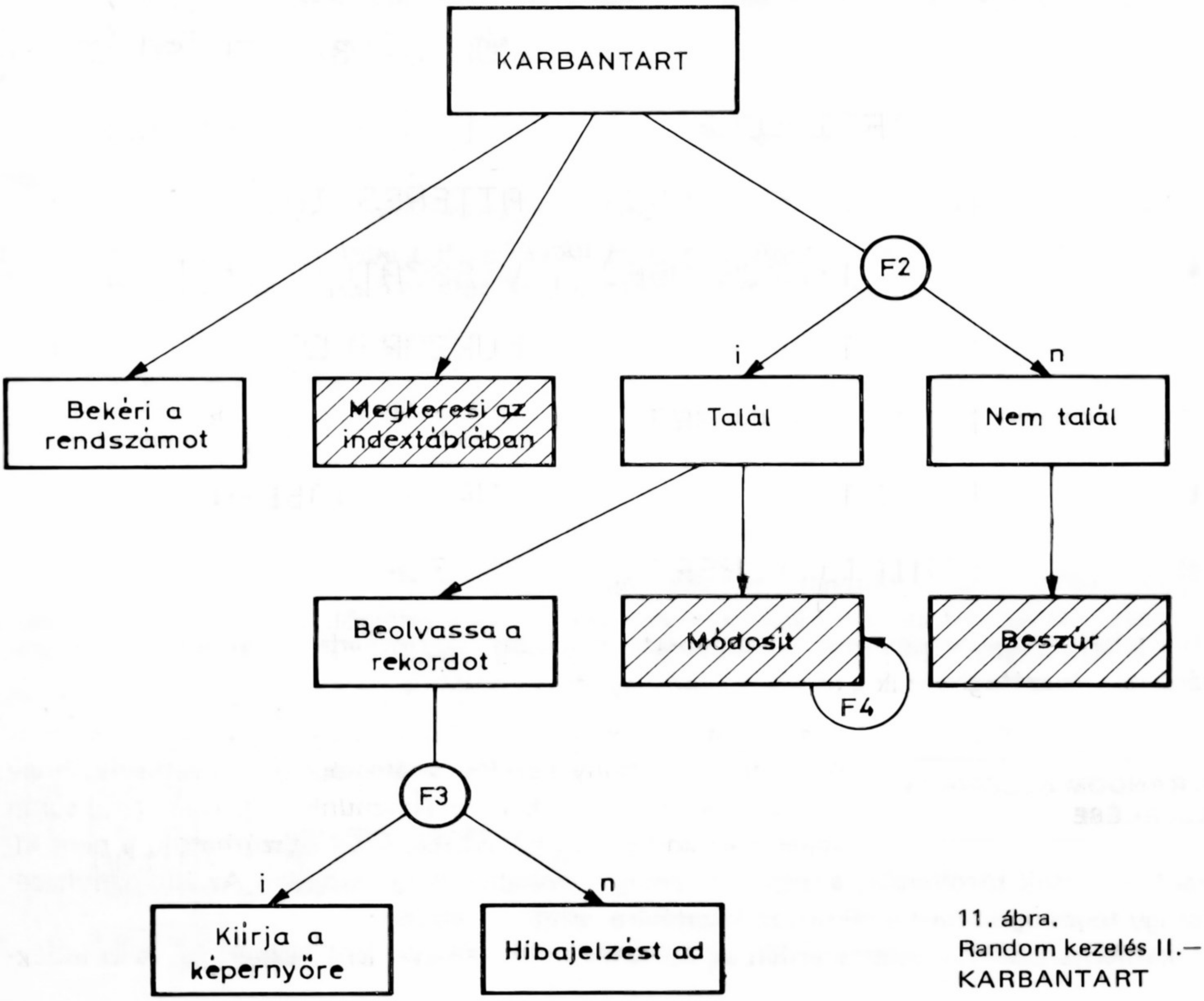
A random állomány kezelési sajátosságaiból következik, hogy ezt az állományt nem kell létrehoznunk. A karbantartás során ugyanis az addig nem létező rekordok beszúrhatók, a nem kívánt rekordok törölhetők, a meglévők pedig olvashatók és felülírhatók. Az állománykezelés így teljes egészében a rekordok kezelésére vezethető vissza.

A karbantartásnak értelemszerűen az indextábla betöltésével kell kezdődnie, és az index-

tábla kimentésével kell végződnie. Kimentéskor természetesen az új, módosított indextábla kerül a lemezre, felülírva a karbantartás előtt lemezen lévő állományt, az indextáblát (10. ábra).



10. ábra. Random kezelés I.



11. ábra.
Random kezelés II.—
KARBANTART

A karbantartást mindaddig folytathatjuk, amíg le nem állítjuk egy végjel átállításával – azaz az F1 feltétel mindaddig teljesül, amíg a VEGE változó értéke logikai „igaz" nem lesz.

Maga a karbantartás két fő funkcióból áll: a meglévő rekordok módosításából és az új rekordok beszúrásából (11. ábra).

A feltételek:

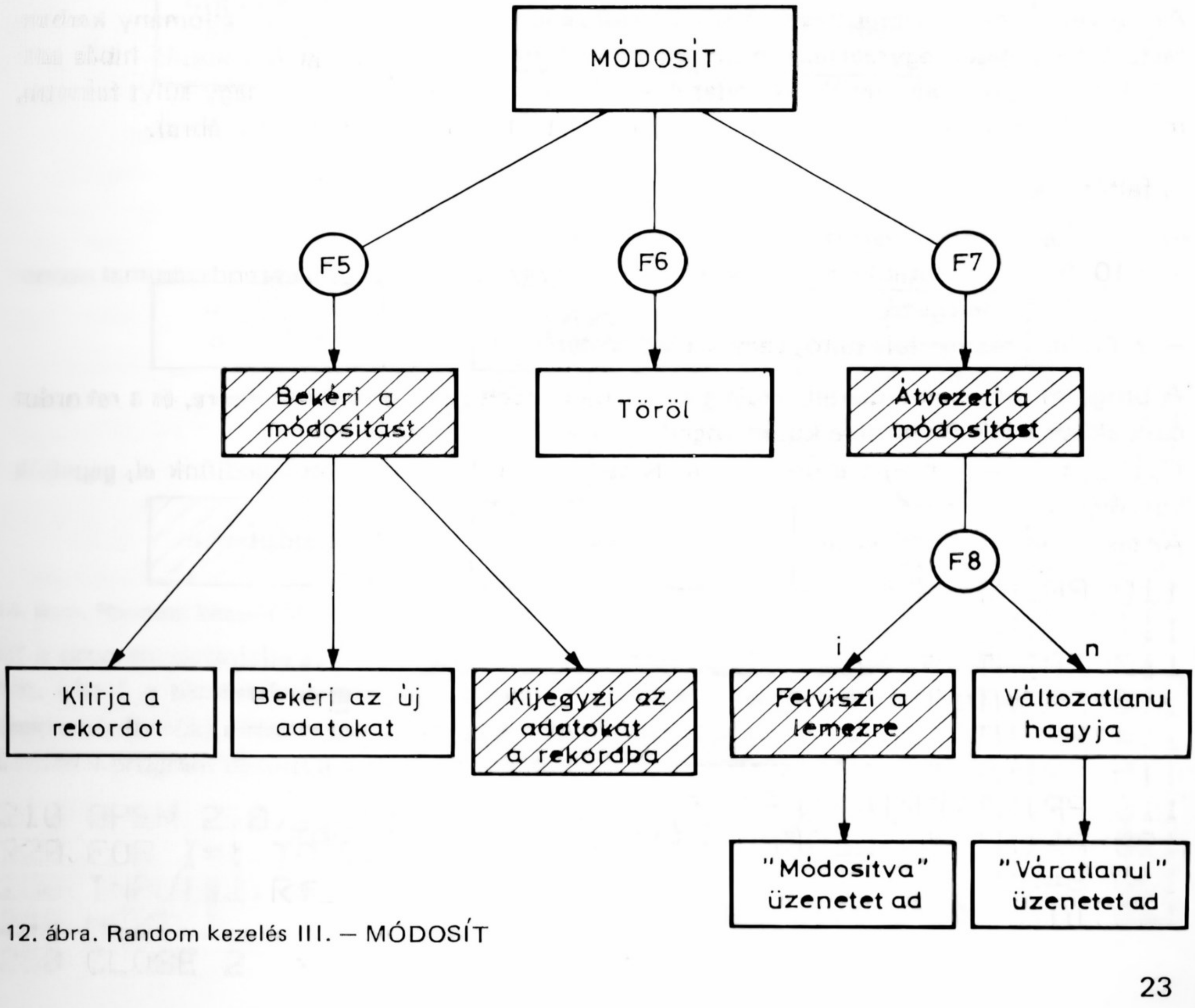
– F2: ha a rendszám szerepel az indextáblában;
– F3: ha az olvasás sikeres volt, azaz a H hibakód <20;
– F4: ha a módosítás nem ért véget, azaz a V$ válasz = "M" vagy "T".

Látható, hogy a karbantartásnak beépített tartozéka a lekérdezés. Ha egy létező rekordot nem módosítunk, és nem is törlünk, akkor automatikusan ez valósul meg.

A módosítás vagy a rekord adatainak megváltoztatását vagy változatlanul hagyását, vagy a teljes rekord törlését jelenti. A módosítás egy rekordra többször is végrehajtható, mindaddig, amíg a módosítást befejezettnek nem tekintjük (12. ábra).

A feltételek:

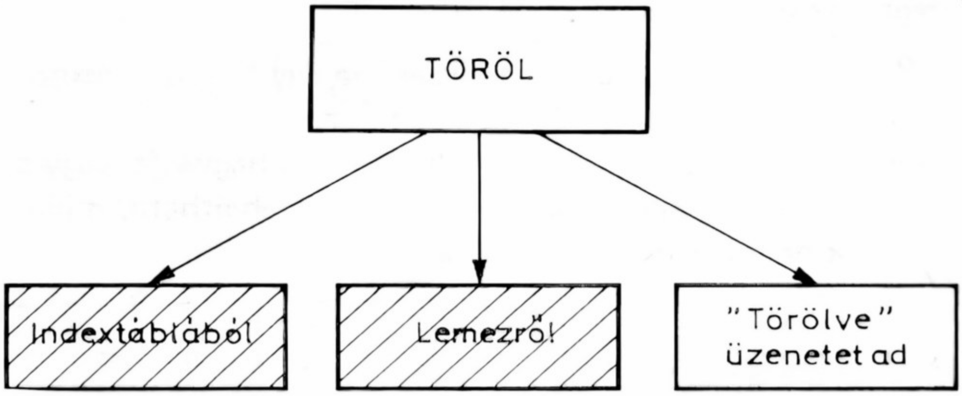
– F5: ha a rekord módosítandó, azaz a V$ válasz = "M";
– F6: ha a rekord törlendő, azaz a V$ válasz = "T";



12. ábra. Random kezelés III. – MÓDOSÍT

– F7: ha a rekord nem változtatandó meg vagy a módosítása befejeződött, azaz a V$ válasz = "*";
– F8: ha a beolvasott rekord nem azonos a rekord aktuális tartalmával.

A program tehát a lemezen lévő rekordot csak akkor írja felül, ha a beolvasott rekord tartalmát megváltoztattuk.
A törlésről már tudjuk, hogy egyszerű, így nem lepődünk meg, ha a program tervében is az (13. ábra).



13. ábra. Random kezelés IV. – TÖRÖL

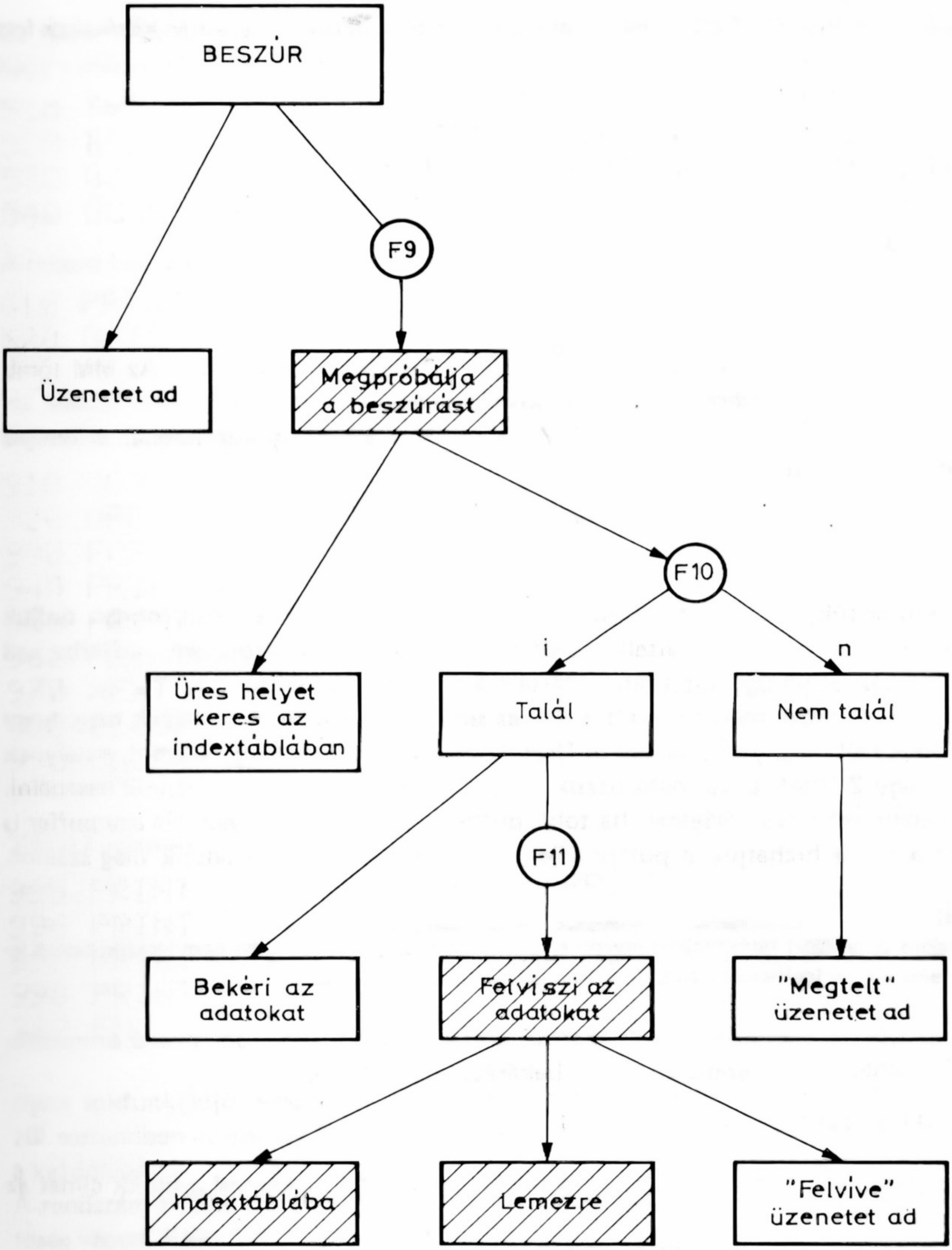
Az új rekordok felvitele beszúrással valósul meg. Minthogy a random állomány karbantartása lényegesen egyszerűbb, mint a soros állományé, az állományba kerülő hibás adatok könnyedén kijavíthatók. Az adatellenőrzésre így nem kell olyan nagy súlyt fektetni, mint a soros felvitelnél. Bizonyos óvatosság azért itt sem mellőzhető (14. ábra).

A feltételek:

– F9: ha a rekord beszúrható, azaz a V$ válasz = "I";
– F10: ha az indextáblában van szabad hely, azaz van csupa csillag rendszámmal azonosított bejegyzés;
– F11: ha a rekord felvihető, vagyis a V$ válasz = "I".

A program tehát lehetőséget nyújt a hibásan begépelt rekord újragépelésére, és a rekordot csak akkor viszi fel, ha erre külön engedélyt kap.
Most pedig lássuk magát a programot. Kicsit hosszú lesz, de ne csüggedjünk el, gépeljük be! Megéri.
Az első lépés a bejelentkezés:

```
110 PRINT "▯"
111 PRINT
112 PRINT "    ▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄"
113 PRINT "    ▌ MENETTELJESITMENYEK █"
114 PRINT "    ▌     KARBANTARTASA    █"
115 PRINT "    ▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀"
116 PRINT:PRINT:PRINT
120 PRINT "    LEGFELJEBB 30 GEPKOCSIRA"
130 S=30
140 DIM R$(S),T(S),B(S)
```



14. ábra. Random kezelés V. — BESZÚR

Itt a program definiálja az indextábla tárbeli formáját, a három, egyenként 30 elemű tömböt. (Az S a tömbméret, jelenleg 30; az R$ a rendszámok, a T a sávok (track) és a B a szektorok (block) címének tárolására alkalmazható.)

Ezután a program betölti a lemezről az indextáblát:

```
210 OPEN 2,8,2,"INDEXTABLA,SEQ,READ"
220 FOR I=1 TO S
230 INPUT#2,R$(I),T(I),B(I)
240 NEXT I
250 CLOSE 2
```

Majd előállítja a hónapok HO$ neveit, amelyeket az adatok bekérésekor és kiírásakor fog használni:

```
310 DIM HO$(12),KM(12),MM(12)
320 DATA "JAN","FEB","MAR","APR"
330 DATA "MAJ","JUN","JUL","AUG"
340 DATA "SEP","OKT","NOV","DEC"
350 FOR I=1 TO 12
360 READ HO$(I)
370 NEXT I
```

A KM tömböt a havi kilométerbeni teljesítmények tárolására definiáltuk. Az MM tömb ugyanezen adatok kimentésére szolgál a módosítás során.
A karbantartás előkészítésének utolsó lépése a random állomány kezeléséhez szükséges parancscsatorna és puffer megnyitása:

```
380 OPEN 15,8,15
390 OPEN 5,8,5,"#"
```

Mint már említettük, a blokk tartalma beolvasáskor egy pufferba kerül, onnan tudjuk azután leolvasni a rekordot. Felvitelkor pedig megfordítva: a rekordot egy pufferba kell beírnunk, a lemezkezelő ugyanis a puffer tartalmát viszi fel a blokkra.
Ezt a puffert kell most megnyitnunk a 390-es sorban. A # jellel határozzuk meg, hogy nem állományt kell megnyitni, hanem puffert. Utána megadhatunk egy számot, amelynek értéke 0, 1 vagy 2 lehet. Ezzel határozzuk meg, hogy melyik puffert kívánjuk használni. Ennek általában akkor van értelme, ha több pufferra is szükségünk van. Ha egy puffer is elég, akkor a gépre bízhatjuk a puffer kijelölését azzal, hogy nem adunk meg számot.

FIGYELEM!
Magát a random állományt nem szabad megnyitnunk, hiszen ez fizikailag soha nem létezik, csak a lemezen lévő rekordjaink logikai összességét jelenti.

A megnyitás után következik a karbantartás, amely a karbantartandó rekord azonosítójának, azaz a gépkocsi R$ rendszámának a bekérésével kezdődik:

```
410 GOSUB 1010
```

és ha a program nem kap leállító jelet, a rendszám alapján megkeresi a blokk címét az indextáblában:

```
420 IF VEGE THEN GOTO 910
430 RR$=RS$
440 GOSUB 1510
```

Itt RR$ a keresett rendszám. Ha ez az indextáblában nem található meg, akkor meghívja a program a beszúrást végrehajtó szubrutint:

```
450 IF TALAL THEN GOTO 510
460 GOSUB 4510
470 GOTO 610
```

Ha viszont szerepel a rendszám az indextáblában, akkor beolvassa a rekordot, majd meghívja a módosító szubrutint:

```
510 T=T(N)
520 B=B(N)
530 GOSUB 3510
540 GOSUB 5010
```

A rekord karbantartása után a program áttér a következő rekord karbantartására:

```
610 PRINT "□"
620 GOTO 410
```

Ha leállító jelet adtunk meg, a karbantartást befejezi a program, majd felülírással kimenti a lemezre az indextáblát:

```
910 CLOSE 5
920 OPEN 2,8,2,"@:INDEXTABLA,SEQ,WRITE"
930 FOR I=1 TO S
940 PRINT#2,R$(I)
950 PRINT#2,T(I)
960 PRINT#2,B(I)
970 NEXT I
980 CLOSE 2
990 CLOSE 15
```

Végül kijelentkezik és leáll:

```
995 PRINT "□":PRINT:PRINT:PRINT
996 PRINT "    ▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄▄"
997 PRINT"   ■ KARBANTARTAS BEFEJEZVE ■"
998 PRINT "    ▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀"
999 END
```

Ilyen sokfunkciójú program már természetes, hogy számos szubrutint használ. Ezek közül sorrendben az első az RS$, a rendszám bekérése, amely minden karbantartó menetnek a kezdőlépése.

A rendszám betűit és számait külön-külön kell megadnunk, a kényelmesebb ellenőrizhetőség végett. Először a BE$ betűket.

```
1010 PRINT "□":PRINT:PRINT
1020 PRINT "**■ RENDSZAM?"
1030 INPUT "      BETU=  **■■■■■";BE$
1040 IF BE$="**" THEN VEGE=1: GOTO 1199
1050 IF LEN(BE$)<>2 THEN GOTO 1030
1060 LB$=LEFT$(BE$,1)
1070 RB$=RIGHT$(BE$,1)
1080 IF "A">LB$ORLB$>"Z" THEN GOTO 1030
1090 IF "A">RB$ORRB$>"Z" THEN GOTO 1030
```

Ha a betűk helyett csillagokat adunk meg, a karbantartás véget ér. Ha nem, a szubrutin formális szempontból ellenőrzi a rendszám betű részét.
Ezután adhatjuk meg a rendszám SZ$ számrészét. A szubrutin ezt is ellenőrzi:

```
1100 INPUT "        SZAM=  0000■■■■■■";SZ$
1110 IF LEN(SZ$)<>4 THEN GOTO 1100
1120 FOR I=1 TO 4
1130 SI$=MID$(SZ$,I,1)
1140 IF "0">SI$ORSI$>"9" THEN GOTO 1100
1150 NEXT I
1160 RS$=BE$+SZ$
1170 VEGE=0
1199 RETURN
```

A rendszám keresése az indextáblában egyszerű rendezetlen soros keresés; ugyanis az esetleges módosítások miatt még akkor sem számíthatunk arra, hogy az indextábla rendezett, ha kezdetben, a beolvasáskor az volt:

```
1510 TALAL=1
1520 FOR N=1 TO S
1530 IF R$(N)=RR$ THEN GOTO 1599
1540 NEXT N
1550 TALAL=0
1599 RETURN
```

A keresésre két esetben van szükség: ha rekordot akarunk beolvasni az állományból, vagy ha új rekordot akarunk oda beszúrni. Az előbbi esetben a rendszámot, az utóbbiban pedig az első üres helyet keressük az indextáblában.
Az adatbekérésre viszont csak akkor kerül sor, ha új rekordot akarunk felvinni, azaz beszúrni. A havi KM menetteljesítményeket külön-külön kell megadnunk minden egyes hónapra. A szubrutin numerikus és nagyságrendi ellenőrzést hajt végre:

```
2010 PRINT "□":PRINT
2020 PRINT "   ■GEPKOCSI■:  ";RS$
2030 PRINT
2040 PRINT "■■ MENETTELJESITMENYEK:"
2050 PRINT
2070 FOR I=1 TO 12
2080 PRINT "- ";HO$(I);"= ";
2090 KM=0 : INPUT KM
2100 IF 0>KM OR KM>9999 THEN GOTO 2080
2110 KM(I)=KM
2120 NEXT I
2199 RETURN
```

Az adatkiírásra többször is szükség van. A rekord beolvasása és minden egyes módosítása után a program megjeleníti a rekordtartalmat a képernyőn:

```
2510 PRINT "□":PRINT
2520 PRINT "  GEPKOCSI:  ";RS$
2530 PRINT
2540 PRINT " MENETTELJESITMENYEK:"
2550 PRINT
2560 EV=0
2570 FOR I=1 TO 12
2580 KM=KM(I)
2590 KK$=RIGHT$("        "+STR$(KM),5)
2600 PRINT "- ";HO$(I);"= ";KK$
2610 EV=EV+KM
2620 NEXT I
2630 EV$=RIGHT$("         "+STR$(EV),6)
2640 PRINT "----------------"
2650 PRINT "= EV =";EV$
2699 RETURN
```

Sőt, kiírja a menetteljesítmények éves EV$ összegét is, noha az nem része a rekordnak. A lemezrevitelre beszúrás és módosítás esetén kerül sor. A szubrutin először felviszi a pufferba a rekordot, azaz az R$ rendszámot és a 12 havi KM teljesítményadatokat:

```
3010 PRINT#5,RS$
3011 FOR I=1 TO 12
3012 PRINT#5,KM(I)
3013 NEXT I
3020 GOSUB 9010
```

A puffert tehát ugyanúgy használja, mintha egyszerű soros állomány lenne. A felvitel sikerességét a hibarutin meghívásával ellenőrzi.

Megjegyezzük, hogy a rendszám felvitele elvileg felesleges, hiszen szerepel az indextáblában; de ha a rendszámot a rekordban is tároljuk, a későbbi beolvasások során ellenőrizhetjük, hogy valóban a keresett rekordot olvastuk-e be.

A szubrutin ezután lefoglal egy blokkot a felírandó rekord számára:

```
3030 PRINT#15,"B-A:"0;T;B
```

Ez a parancscsatornára kiadott állománykezelő információk hatására megy végbe.

Az idézőjelek között a BLOCK–ALLOCATE kulcsszó rövidítése szerepel. Ezzel rendeljük el a blokk lefoglalását, „allokálását".

Ezután meg kell adnunk a lemezmeghajtó számát, ami a 1541-es lemezegységen mindig 0. Majd a T sávcím és a B szektorcím (az eredeti COMMODORE-szóhasználattal track és block address) következik. Ezek értékét az indextáblából kapjuk.

Megjegyezzük, hogy a fenti állománykezelő utasításban semmilyen más szeparátorjel nem használható, mint a példában bemutatottak.

A blokk lefoglalása után a szubrutinnak mindig le kell kérdeznie a parancscsatornát. Ha ugyanis a blokk szabad volt, akkor a lefoglalás sikeres lesz, és H=0 hibakódot kapunk. Foglalt blokk esetén a lefoglalás sikertelen, és a H hibakód értéke 65 lesz:

```
3040 GOSUB 9010
3050 IF H=65 THEN T=TT:B=SS:GOTO 3030
```

Amint bizonyára emlékezünk rá, a parancscsatorna a H hibakód és a H$ hibaüzenet mellett egy TT sávcímet és egy BB szektorcímet is megad. Ez általában a hibás blokk címe, kivéve a blokk lefoglalásának esetét.

Blokkfoglalás, azaz BLOCK–ALLOCATE után a parancscsatorna a legközelebbi szabad blokk címét adja meg, ha a lefoglalandó blokk már foglalt volt.

A „legközelebbi" azt jelenti, hogy a kérdéses blokkot követő, azaz magasabb blokkcímeken elhelyezkedő blokkok közül az első szabad blokk címét kapjuk meg, ha van ilyen. Minthogy a szabad blokk keresése mindig csak előrefelé, a magasabb blokkcímek felé halad, tehát visszafelé keresés nincs, előfordulhat, hogy a megadott címnél csak kisebb című blokkok állnak üresen. Ekkor – az üres blokkok ellenére – a BLOCK–ALLOCATE nem talál szabad blokkot, és **NO BLOCK, TT=0, BB=0** hibajelzést kapunk. A program ezt úgy védi ki, hogy a keresést mindig az indextáblából vett 1-es sáv 1-es szektor címről, vagyis a lemez elejéről indítja. Sikertelen blokkfoglalás esetén a szubrutin a műveletet mindaddig megismétli a parancscsatorna által megadott blokkra, amíg a blokk lefoglalása sikeres nem lesz.

Itt látszólag végtelen ciklust generáltunk. Valójában az első sikertelen blokkfoglalási kísérlet esetén a második sikeres lesz, hiszen ezt éppen az első által megadott szabad helyre hajtjuk végre. Ha pedig az első foglalás azért nem talált szabad blokkot, mert például megtelt a lemez, akkor a ciklus el sem jut a második menetig, mert a program már a hibarutinban a 9091 STOP utasításnál **NO BLOCK** hibaüzenettel leáll.
Sikeres blokkfoglalás esetén a program a puffer tartalmát felviszi a blokkra:

```
3060 PRINT#15,"B-W:"5;0;T;B
3070 GOSUB 9010
```

Ehhez is állománykezelő információkat kell megadni, szükség van tehát a parancscsatornára. (Az idézőjelek között itt is rövidítés áll: a BLOCK–WRITE, azaz a blokkírás kulcsszóé.)
Ezután meg kell adnunk a puffer számára lefoglalt adatcsatornát (ami a példánkban 5 volt), majd a lemezmeghajtó számát (0), a sávcímet (T) és a szektorcímet (B).
Az utasítás hatására a lemezkezelő felviszi a megadott csatornához rendelt puffer tartalmát (254 bájtot) a megadott sáv megadott szektorára, akár szabad volt a kérdéses blokk, akár foglalt; akár a random állományhoz tartozik, akár valamelyik más állományhoz, programhoz vagy a lemeznyilvántartáshoz.
Biztonsági okokból tehát csak szabad, azaz frissen lefoglalt blokkra célszerű írni. A puffer tartalmát a lemezkezelő minden vizsgálat nélkül viszi fel a blokkra; a programozó dolga, hogy előzőleg azzal töltse fel a puffert, amit a lemezre kíván írni.
Végül bejegyzi a szubrutin az indextáblába a rekord adatait, az RS$ rendszámot, a T sávcímet és a B szektorcímet:

```
3080 R$(N)=RS$
3090 T(N)=T
```

```
3100 B(N)=B
3199 RETURN
```

A bejegyzés N helyét az indextáblában végrehajtott keresés határozza meg.
A megtalált blokkról külön szubrutin olvassa be a rekordokat. Az első lépésben betölti a blokkot a pufferba:

```
3510 PRINT#15,"B-R:"5;0;T;B
3520 GOSUB 9010
```

A puffer feltöltése a parancscsatornára kiadott információk hatására következik be.
Az idézőjelben a BLOCK—READ, azaz a blokkolvasás kulcsszó rövidítése látható. Ezzel határozzuk meg, hogy egy blokk olvasása hajtandó végre. Természetesen meg kell adnunk, hogy a beolvasás honnan történjék, azaz a feltöltendő puffer csatornaszámát (5), a lemezmeghajtó számát (0), a beolvasandó blokk sávcímét (T) és szektorcímét (B). Ennek hatására a lemezkezelő a megadott blokk tartalmával feltölti a megadott puffert. A beolvasás sikerességéről illik meggyőződni a parancscsatorna lekérdezésével.
A szubrutin ezután a már feltöltött blokkból beolvassa a rekordot, mintha a blokk egy soros adatállomány lenne:

```
3530 INPUT#5,RR$
3531 FOR I=1 TO 12
3532 INPUT#5,MM(I)
3533 KM(I)=MM(I)
3534 NEXT I
3540 GOSUB 9010
```

A 12 teljesítményadatot az MM tömbben megőrzi, hogy a program később ellenőrizhesse, ténylegesen módosult-e a beolvasott rekord. A biztonság okáért ellenőrzi a szubrutin, hogy a rekordban tárolt RR$ rendszám azonos-e az indextáblában szereplő RS$ rendszámmal, azaz hogy a blokk címe nem került-e tévesen az indextáblába:

```
3550 IF RR$=RS$ THEN GOTO 3699
```

Ha mindent rendben talál, a szubrutin befejeződik; ha nem, akkor a rekord tartalmát megjeleníti a képernyőn, megfelelő hibaüzenettel kiegészítve:

```
3560 GOSUB 2510
3570 PRINT
3580 PRINT "RENDSZAM <";RR$;"> NEM AZONOS!"
3590 PRINT:PRINT "BEGEPELENDO:"
3600 PRINT:PRINT "■CLOSE5:CLOSE15■"
3610 PRINT:PRINT "VAGY:"
3620 PRINT:PRINT "■CONT■"
3630 STOP
3699 RETURN
```

Ilyenkor választhatunk, hogy a hibát olyan súlyosnak tekintjük-e, amiért le kell zárni a feldolgozást, vagy tovább engedjük.

A törlő szubrutin végrehajtására csak akkor kerül sor, ha a karbantartáskor a törlés ("T") funkciót választjuk:

```
4010 R$(N)="******"
4020 PRINT#15,"B-F:"0;T(N);B(N)
4030 GOSUB 9010
4040 T(N)=1
4050 B(N)=1
4099 RETURN
```

A szubrutin az indextáblából úgy törli az R$ rendszámot, hogy csillagokkal írja felül. (Ezek jelzik a táblában az üres helyeket.) A sáv- és a szektorcímet csak akkor állítja vissza a kezdőértékre, ha a blokk felszabadítását már végrehajtotta.

Megjegyezzük, hogy a blokkcímet nem feltétlenül kell visszaállítani: meghagyhatnánk az indextáblában a törölt blokk címét is, hiszen az már szabad. Így a legközelebbi rekordfelvitel során megkeresvén az indextáblában az első szabad helyet — ami most a törölt rekord helye —, ott azonnal egy szabad blokk címét találná. Ez a megoldás azonban csak akkor biztonságos, ha a törlést mindig közvetlenül követi az új felvitel, és a lemezen nincs több állományunk. Ha ugyanis azok valamely blokkja elfoglalja a törléskor felszabadított blokkot, előfordulhat az az eset, amiről már szóltunk; hogy nem találunk szabad blokkot, noha a törölt blokk helye előtt még van szabad blokk. Ezért jobb biztosítani, hogy a keresés mindig a lemez elejéről induljon.

A rekord fizikai törlését a lemezről a parancscsatornára kiadott BLOCK—FREE kulcsszó hatására hajtja végre a lemezkezelő, mégpedig úgy, hogy felszabadítja az állomány számára lefoglalt blokkot. Lemezkezelő információként meg kell adnunk a kulcsszót, vagy annak rövidítését (B—F), a törlendő blokk sávcímét (T) és szektorcímét (B).

A felszabadítás ténylegesen azt jelenti, hogy a lemezkezelő a blokk foglaltságát jelző bitet a lemeztérképen foglaltról szabadra állítja. (A művelet sikerességét a hibarutin ellenőrzi.)

Az új rekordot beszúró szubrutin végrehajtására akkor kerül sor, ha a karbantartáskor megadott rendszám nem található az indextáblában. Ilyenkor először megfelelő üzenetet kapunk:

```
4510 PRINT "■":PRINT:PRINT
4520 PRINT "  ■GEPKOCSI■: ";RS$:PRINT
4530 PRINT "  ■NINCS NYILVANTARTVA■"
4540 PRINT
4550 PRINT "■*■ BESZURHATO? ";
4551 INPUT "=I/N=  N■■■■";V$
4560 IF V$="N" THEN GOTO 4899
4570 IF V$<>"I" THEN GOTO 4550
```

Ekkor még ellenőrizhetjük, hogy a megadott rendszám valóban új-e, tehát rekordja beszúrható, vagy csak téves begépelés miatt nem található az indextáblában.

Ha a rekord beszúrása mellett döntünk, akkor a szubrutin keres egy üres, azaz rendszám helyett csillagokat tartalmazó helyet az indextáblában:

```
4580 RR$="******"
4590 GOSUB 1510
```

Ha ilyet nem talál, megtelt jelzést ad a képernyőn:

```
4600 IF TALAL THEN GOTO 4710
4610 U$="A TABLA MEGTELT"
4620 GOSUB 6010
4630 GOTO 4899
```

Ha talál szabad helyet, bekéri a rekord adatait:

```
4710 GOSUB 2010
```

Az adatok begépelése után még ellenőrizthetjük, hogy nem követtünk-e el gépelési hibát. Ha a rekord felvitelét megtiltjuk, akkor a szubrutin nem viszi fel a lemezre:

```
4720 PRINT
4730 PRINT "**■ FELVIHETO? ";
4731 INPUT "=I/N=  N■■■■";V$
4740 IF V$="N" THEN GOTO 4899
4750 IF V$<>"I" THEN GOTO 4730
```

Ilyenkor a rekordot egy újabb, most már remélhetőleg hibátlan beszúrási menettel vihetjük fel.

Ha viszont a rekordot felvihetőnek minősítettük, akkor az indextábla alapján beállítja a szubrutin a blokkcímet, majd felviszi a rekordot a lemezre, és ennek megfelelő üzenetet ír ki a képernyőre:

```
4780 T=T(N)
4790 B=B(N)
4800 GOSUB 3010
4810 U$="FELVIVE"
4820 GOSUB 6010
4899 RETURN
```

A módosítás összetett funkció. Magában foglalja a rekord adatainak megváltoztatását, a teljes rekord törlését vagy változatlanul hagyását. Először a szubrutin megjeleníti a módosítandó rekordot a képernyőn:

```
5010 GOSUB 2510
```

Ekkor dönthetünk, hogy módosítjuk, töröljük, vagy változatlanul hagyjuk a rekordot:

```
5020 PRINT
5030 INPUT "**■ MODOSITANDO/TORLENDO =M/T=  *■■■■";V$
5040 IF V$="M" THEN GOTO 5060
5050 IF V$="T" THEN GOTO 5410
5055 IF V$="*" THEN GOTO 5210
5056 PRINT "]";
5059 GOTO 5030
```

A funkciókat az M vagy a T betű, illetve a * megadásával választhatjuk ki. Minden más válasz a kérdés megismétlésével jár.

Ha a módosítást választottuk, a szubrutin sorra megjelöli egy nyíllal a módosítandó adatokat. Egyszerűen RETURN választ adunk, ha valamelyiket nem kívánjuk módosítani. Módosítási szándék esetén gépeljük be az új adatot, és csak ez után nyomjuk meg a RETURN gombot! A begépelt adatot külön szubrutin vezeti be a teljesítményadatok közé:

```
5060 PRINT "[IIIIIIIIIIIIIIIII]"
5070 FOR I=1 TO 12
5080 PRINT "▮▮▮▮▮▮▮▮▮▮▮▮▮▮▮";
5090 M$=STR$(0):INPUT "← ";M$
5100 GOSUB 5510
5110 NEXT I
```

A módosítás után ellenőrizhetjük annak helyességét, vagy helybenhagyjuk a módosított változatot, vagy újra módosítunk:

```
5120 PRINT "▮▮▮▮▮▮"
5140 GOTO 5010
```

A módosítás többször is megismételhető.

Ha nem akarunk tovább módosítani, akkor a szubrutin kérdésére csillaggal válaszolunk. Ekkor a szubrutin megnézi, hogy a teljesítményadatok jelenlegi KM értéke megegyezik-e az MM-be eredetileg beolvasottal:

```
5210 FOR I=1 TO 12
5220 IF KM(I)<>MM(I) THEN GOTO 5310
5230 NEXT I
```

Ha egyezik (mert a rekordot csak lekérdeztük, vagy mert a rekordot tévedésből módosítottuk, majd a hibát észrevéve a téves módosítást helyreigazítottuk), akkor a szubrutin a rekordot változatlanul hagyja, és a blokkba nem írja vissza:

```
5240 U$="VALTOZATLAN"
5250 GOSUB 6010
5260 GOTO 5499
```

A rekord változatlanul hagyásáról a szubrutin US üzenettel tájékoztat. Ha viszont a beolvasott érték eltér a rekord aktuális értékétől, akkor nyilvánvalóan módosítás történt. Ilyenkor a szubrutin a rekordot felírja a lemezre:

```
5310 T=T(N)
5320 B=B(N)
5330 GOSUB 3010
5340 U$="MODOSITVA"
5350 GOSUB 6010
5360 GOTO 5499
```

A rekord törlése esetén, azaz ha a karbantartáskor a "T" funkciót választottuk, akkor a szubrutin behívja a törlő szubrutint:

```
5410 GOSUB 4010
5420 U$="TOROLVE"
5430 GOSUB 6010
5499 RETURN
```

A módosításokat át kell vezetni a módosítandó rekordba; ez a tulajdonképpeni módosítás. Végrehajtására külön szubrutin van, mely először is ellenőrzi az M$ módosító adatot:

```
5510 L=LEN(M$)
5530 IF 1>L OR L>4 THEN GOTO 5599
5540 FOR J=1 TO L
5550 L$=MID$(M$,J,1)
5560 IF L$<"0" THEN GOTO 5599
5570 IF L$>"9" THEN GOTO 5599
5580 NEXT J
```

A formailag hibátlan M$ adatot áttölti a módosítandó rekord megfelelő KM helyére:

```
5590 KM(I)=VAL(M$)
5599 RETURN
```

Formailag hibás M$ adattal a módosítást nem hajtja végre. Az adatbekérés azért karakteres formájú, hogy a módosításhoz megjelenített rekordtartalmat a hibás válaszadás esetén a képernyőre íródó gépi hibaüzenet ne bonthassa meg.

A karbantartással kapcsolatos U$ üzenetek, például IMODOSITVA, FELVIVE, TOROLVE, VALTOZATLAN stb. kiírását külön szubrutin végzi:

```
6010 PRINT
6020 PRINT "          ■";U$;"■"
```

Az U$ üzenet kiírása után a szubrutin időt hagy az üzenetek elolvasására:

```
6030 T1=TI
6040 T2=TI
6050 IF T2-T1<150 THEN GOTO 6040
6099 RETURN
```

Ehhez a szubrutin a gép óráját (TIME) kérdezi le, mindaddig, amíg bizonyos idő (mintegy 2,5 másodperc) el nem telik.

Minden fontosabb lemezművelet után meghív a program egy hibakezelő rutint, amelyben lekérdezi a parancscsatornát. Hiba esetén kiírja az éppen feldolgozás alatt álló rekord RS$ azonosítóját, a rendszámot, valamint a H hibakódot és a lemezkezelő H$ hibaüzenetét:

```
9010 INPUT#15,H,H$,TT,SS
9020 IF H<20 OR (H=65 AND TT<>0) THEN GOTO 9099
9030 PRINT "■":PRINT:PRINT
9040 PRINT "HIBA: ";RS$
9050 PRINT H;" - ";H$
```

Ezután választhatunk, hogy a karbantartást befejezzük-e, vagy folytatjuk:

```
9060 PRINT:PRINT "BEGEPELENDO:"
9070 PRINT:PRINT "▮CLOSE5:CLOSE15▮"
9080 PRINT:PRINT "VAGY:"
9090 PRINT:PRINT "▮CONT▮"
9091 STOP
9099 RETURN
```

Megjegyezzük, hogy a blokk lefoglalása (BLOCK–ALLOCATE) után feltétlenül le kell kérdezni a parancscsatornát, azaz meg kell hívni a lekérdező szubrutint, mert másként nem győződhetünk meg a blokkfoglalás sikerességéről, illetve csak így kapható meg a legközelebbi szabad blokk címe.

Ezzel a program elkészült. Ha begépeltük, mentsük ki RANDOMKEZELES néven, majd indítsuk el. Elsőként AA 1111 rendszámot adjunk meg.

```
? RENDSZAM?
      BETU=? AA
      SZAM=? 1111
```

Minthogy most az indextáblában egyetlen rendszám sincs, a program megkérdezi, hogy az új rekordot beszúrhatja-e.

```
  [illegible]: AA1111

  [illegible]

? BESZURHATO? =I/N=? I
```

Válaszoljunk igennel, s amikor a program az adatokat kéri, mind a 12 hónapra nulla kilométer teljesítményt adjuk meg (lásd 37. oldal felső ábra).

Tegyük ugyanezt még három másik rekorddal is, mondjuk FF 6666, DD 4444 és II 9999 fiktív rendszámokat használva.

A „**Felvihető?**" kérdésre természetesen mindig igenlő választ adjunk, feltéve persze, hogy nem vétettünk gépelési hibát.

Most tehát már van négy rekordból álló random állományunk. Módosítsuk a DD 4444 gépkocsi adatait. Adjuk meg a rendszámot. Ekkor az adatok megjelennek a képernyőn.

Adjunk "M" választ, minthogy módosítani akarunk. Most módosítsuk a januári értéket 0-ról 100-ra!

Ezután fejezzük be a módosítást, vagyis a program kérdésére hagyjuk meg az előre megadott csillagot, és nyomjuk meg a RETURN gombot. (38. oldal.)

A következő lépésben kérdezzük le az FF 6666 gépkocsi adatait. A rendszám megadása után a rekord megjelenik a képernyőn (lista a 39 oldalon).

```
GEPKOCSI: AA1111

* MENETTELJESITMENYEK:

- JAN= ? 0
- FEB= ? 0
- MAR= ? 0
- APR= ? 0
- MAJ= ? 0
- JUN= ? 0
- JUL= ? 0
- AUG= ? 0
- SEP= ? 0
- OKT= ? 0
- NOV= ? 0
- DEC= ? 0

* FELVIHETO? =I/N=? I

      FELVIVE


GEPKOCSI: DD4444

* MENETTELJESITMENYEK:

- JAN=        0
- FEB=        0
- MAR=        0
- APR=        0
- MAJ=        0
- JUN=        0
- JUL=        0
- AUG=        0
- SEP=        0
- OKT=        0
- NOV=        0
- DEC=        0
-----------------
= EV =        0

* MODOSITANDO/TORLENDO =M/T=? M
```

```
[illegible]: DD4444

▒ MENETTELJESITMENYEK:

- JAN=      0← ? 100
- FEB=      0
- MAR=      0
- APR=      0
- MAJ=      0
- JUN=      0
- JUL=      0
- AUG=      0
- SEP=      0
- OKT=      0
- NOV=      0
- DEC=      0
-----------------
= EV =      0

▒ MODOSITANDO/TORLENDO =M/T=? M
```

```
[illegible]: DD4444

▒ MENETTELJESITMENYEK:

- JAN=    100
- FEB=      0
- MAR=      0
- APR=      0
- MAJ=      0
- JUN=      0
- JUL=      0
- AUG=      0
- SEP=      0
- OKT=      0
- NOV=      0
- DEC=      0
-----------------
= EV =    100

▒ MODOSITANDO/TORLENDO =M/T=? *

    [illegible]
```

```
GEPKOCSI: FF6666

> MENETTELJESITMENYEK:

- JAN=      0
- FEB=      0
- MAR=      0
- APR=      0
- MAJ=      0
- JUN=      0
- JUL=      0
- AUG=      0
- SEP=      0
- OKT=      0
- NOV=      0
- DEC=      0
--------------
= EV =      0

> MODOSITANDO/TORLENDO =M/T=? *

      VALTOZATLAN
```

A rekordot hagyjuk változatlanul, azaz a válaszunk csillag legyen, majd adjuk meg az II 9999 rendszámot. Amikor a rekord megjelenik a képernyőn, karbantartási funkcióként adjunk "T" választ, tehát töröljük a rekordot:

```
GEPKOCSI: II9999

> MENETTELJESITMENYEK:

- JAN=      0
- FEB=      0
- MAR=      0
- APR=      0
- MAJ=      0
- JUN=      0
- JUL=      0
- AUG=      0
- SEP=      0
- OKT=      0
- NOV=      0
- DEC=      0
--------------
= EV =      0

> MODOSITANDO/TORLENDO =M/T=? T

      TOROLVE
```

Hívjuk be ismét a már módosított rekordot a DD 4444 rendszám megadásával, és módosítsuk a februári 0 adatot 200-ra:

```
    [illegible]: DD4444

:: MENETTELJESITMENYEK:

- JAN=    100← ?
- FEB=      0← ? 200
- MAR=      0
- APR=      0
- MAJ=      0
- JUN=      0
- JUL=      0
- AUG=      0
- SEP=      0
- OKT=      0
- NOV=      0
- DEC=      0
-----------------
= EV =    100

:: MODOSITANDO/TORLENDO =M/T=? M
```

Újra válasszuk az "M" módosítás funkciót, és változtassuk a márciusi és áprilisi adatokat 0-ról 300-ra, illetve 400-ra:

```
    [illegible]: DD4444

:: MENETTELJESITMENYEK:

- JAN=    100
- FEB=    200
- MAR=    300
- APR=    400
- MAJ=      0
- JUN=      0
- JUL=      0
- AUG=      0
- SEP=      0
- OKT=      0
- NOV=      0
- DEC=      0
-----------------
= EV =   1000

:: MODOSITANDO/TORLENDO =M/T=? *
```

[illegible] — Fejezzük be a módosítást.

Kérjük be az II 9999 rendszámú gépkocsi adatait. **„NINCS NYILVANTARTVA"** üzenetet kapunk, mivel a rekordot már töröltük.

```
RENDSZAM: II9999

NINCS NYILVANTARTVA

 BESZURHATO? =I/N=? N
```

A **„Beszúrható?"** kérdésre adjunk nemleges választ.

Végül kérjük be a BB 2222 rendszámú gépkocsi adatait. Ez sincs nyilvántartva, de most szúrjuk be. Szintén csupa nulla adatot adjunk meg. Ezután állítsuk le a programot úgy, hogy a rendszám bekérésekor betűk helyett két csillagot adunk meg.

Ezzel a karbantartó program fő funkcióit, vagyis a beszúrást, a módosítást, a változatlanul hagyást, a törlést és a lekérdezést teszteltük.

Érdemes persze a programot alaposan próbára tennünk számos más szempontból is. Például: hogyan viselkedik, ha a rendszámnak 3 betűt adunk meg, vagy a számrész nem numerikus? Mit tesz a program, ha betelik az indextábla? Mi történik, ha módosítás funkciót választunk, de a havi adatokat nem módosítjuk? Milyen adat kerül a rekordba, ha az adatbekéréskor semmilyen adatot nem gépelünk be, csak a RETURN gombot nyomjuk meg? Milyen adat kerül a rekordba, ha felvitelkor nem numerikus értéket gépelünk be? Mi történik, ha ugyanezt tesszük módosításkor? Mit csinál a program, ha a feltett kérdéseire nem az előírt betűkkel válaszolunk? És még folytathatnánk tovább a kérdéseket, amelyekre választ csak egy alapos tesztelés adhat.

Ha a tesztelésen túlvagyunk, töltsük be és listázzuk ki a lemez tartalomjegyzékét. Látni fogjuk, hogy abban a random állomány nem szerepel. Ennek okát már korábban ismertettük: a random állomány csak számunkra létezik, a rekordok logikai összességét jelenti.

A random állomány jelenlétére a lemezen csak az utal, hogy a szabad blokkok száma nem egyezik meg az adattárolásra használható blokkok számának (664-nek), valamint a tartalomjegyzékben felsorolt állományok és programok által elfoglalt blokkok összegének a különbségével.

FIGYELEM!

Mivel a lemezen már van random állomány, nem adhatunk ki VALIDATE parancsot. Ennek hatására ugyanis a lemez újraszerveződne, ami a tartalomjegyzékben rögzített állományokhoz nem tartozó blokkok felszabadításával, így a **random állomány megsemmisítésével** járna.

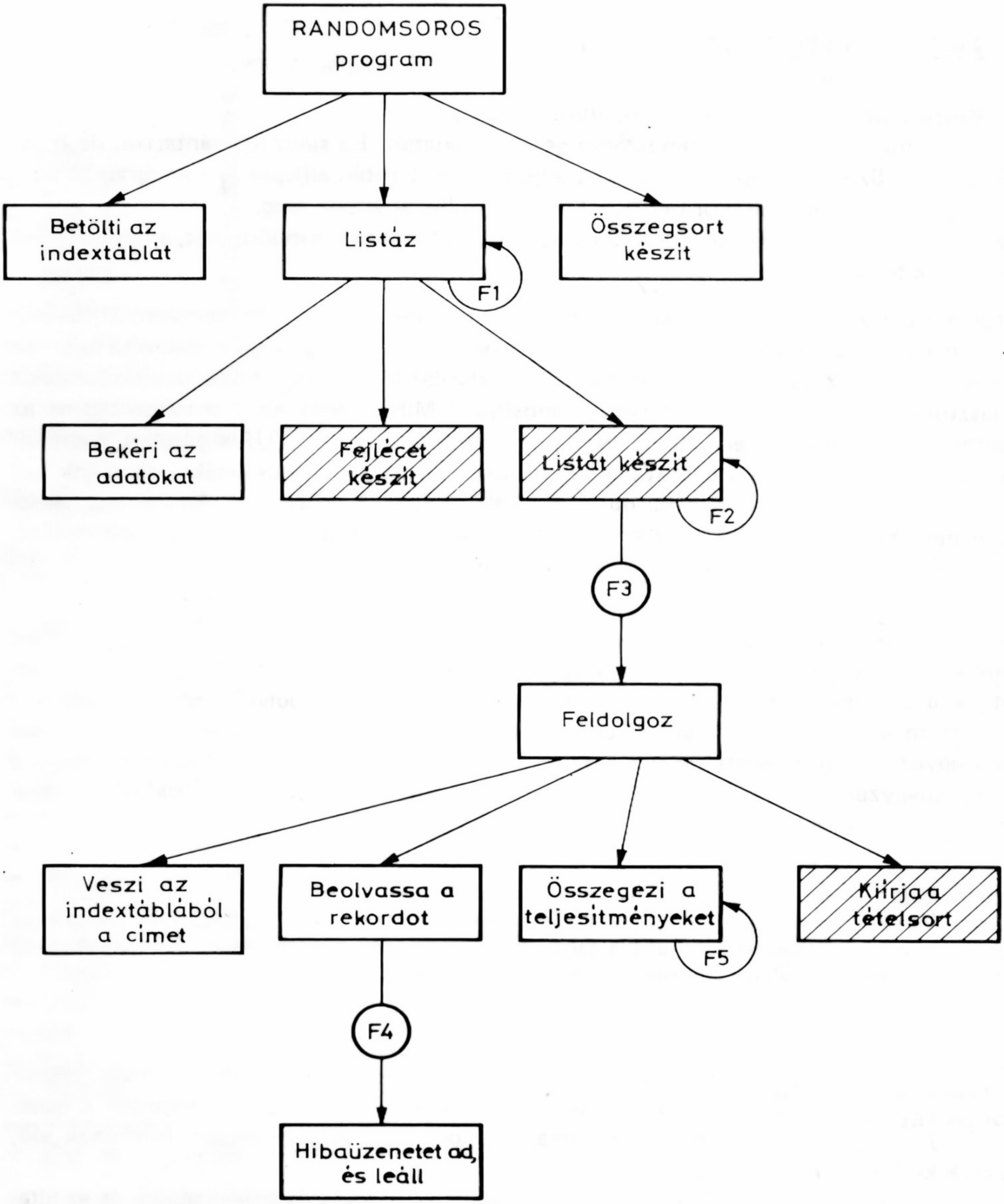
A RANDOM ÁLLOMÁNY SOROS FELDOLGOZÁSA

Az indextáblával kezelt random állomány előnye, hogy rekordjai sorosan is feldolgozhatók, noha maguk a rekordok a lemezen össze-vissza, gyakorlatilag véletlenszerűen fordulnak elő, és csak közvetlenül érhetők el.

A feldolgozás lényege, hogy sorban végigmegyünk az indextábla bejegyzésein, és az általuk meghatározott rekordokat feldolgozzuk, természetesen figyelmen kívül hagyva az

indextábla üres (csillaggal feltöltött) helyeit. Ilyenkor a random állomány rekordjainak soros feldolgozását közvetlen elérések sorozatával valósítjuk meg.

Tekintsünk egy egyszerű mintafeladatot: készítsünk listát az egyes gépkocsik tetszőleges intervallumon belüli összesített menetteljesítményéről, ahol az időintervallum adott hónaptól adott hónapig, de legfeljebb januártól decemberig tarthat (15. ábra):



15. ábra. Random soros feldolgozás

A feltételek:

- F1: ha nincs vége a listázásnak, azaz VEGE="N";
- F2: 1-től 30-ig egyesével;
- F3: ha a kiírandó rekord létezik, vagyis a rendszám az indextáblában nem csupa csillag;
- F4: ha hibás volt az olvasás, tehát a H hibakód > 19;
- F5: a H1 kezdő hónaptól a H2 záró hónapig egyesével.

Látható, hogy a program nemcsak egy listát készít, hanem addig készíti a különböző listákat, amíg le nem állítjuk.

A bejelentkezéssel kezdjük:

```
1 PRINT "♥"
2 PRINT
3 PRINT " MENETTELJESITMENYEK"
4 PRINT "      KILISTAZASA      "
5 PRINT " --------------------"
6 PRINT
7 PRINT
```

Majd definiálja az indextábla R$, T, B tömbjeit, a havi teljesítmények K tömbjét és a kezdőértékeket:

```
10 DIM R$(30),T(30),B(30)
20 DIM K(12)
30 E=3
40 C$="******"
```

Megjegyezzük, hogy a lista a képernyőn jelenik meg, de egyetlen sor, a 30-as megváltoztatásával (vagyis ha az E=3 utasítást kicseréljük az E=4 utasításra) a listázás a nyomtatón történik. Elvileg megoldható az is, hogy a kiírásra használt eszköz egységszámát a billentyűzetről kérjük be. Ezt egy korábbi példánkban meg is tettük.

A program ezután megnyitja a parancscsatornát, a random állomány kezeléséhez szükséges puffert és azt az egységet, amelyre a listázás végrehajtandó:

```
100 OPEN 15,8,15
101 OPEN 3,8,3,"#"
102 OPEN 1,E
```

Majd a már ismert módon betölti az indextáblát:

```
110 OPEN 2,8,2,"INDEXTABLA,SEQ,READ"
120 FOR I=1 TO 30
130 :   INPUT#2,R$(I),T(I),B(I)
140 NEXT I
199 CLOSE 2
```

Bekéri a H1–H2 listázási intervallumot, és ellenőrzi is, hogy értelmes-e. (Ha nem az, a bekérést megismétli.)

```
210 INPUT " MELYIK HONAPTOL = ";H1
220 IF H1<1 OR H1>12 THEN GOTO 210
230 :  PRINT
240 :  INPUT " MELYIK HONAPIG  = ";H2
250 :  IF H2<1 OR H2>12 THEN GOTO 240
260 :  PRINT
270 IF H2<H1 THEN GOTO 210
280 :  H1=INT(H1)
281 :  H1$=RIGHT$(STR$(H1),2)
290 :  H2=INT(H2)
291 :  H2$=RIGHT$(STR$(H2),2)
```

A hónapszámok karakteressé alakítása csak a nyomtatási formátum egységessége érdekében szükséges.

Elkészíti a lista fejlécét:

```
310 :  PRINT#1,"▢"
320 :  PRINT#1
330 :  PRINT#1," ♥♥ MENETTELJESITMENYEK ♥♥"
340 :  PRINT#1," ----------------------------"
350 :  PRINT#1," ";H1$;". HONAPTOL ";
360 :  PRINT#1," ";H2$;". HONAPIG"
370 :  PRINT#1," ----------------------------"
380 :  PRINT#1
```

Kinullázza a mindösszesen MM gyűjtőjét:

```
410 :  MM=0
```

Eddig tartott a listázás előkészítése.

A tulajdonképpeni feldolgozás most kezdődik: a program végigmegy a teljes indextáblán. A C$ csillaggal feltöltött bejegyzéseket figyelmen kívül hagyja. Amelyikben azonban rendszámot talál, ahhoz beolvassa a megfelelő rekordot:

```
510 :  FOR I=1 TO 30
520 :  :  IF R$(I)=C$ THEN GOTO 699
530 :  :  RR$=R$(I)
540 :  :  T=T(I)
550 :  :  B=B(I)
560 :  :  GOSUB 1010
```

A beolvasott rekordból veszi a teljesítmények K adatait, és összegezi őket a megadott időintervallumon belül:

```
570 :  :  M=0
580 :  :  FOR J=H1 TO H2
590 :  :  :  M=M+K(J)
599 :  :  NEXT J
```

Kiírja az adott gépkocsinak az adott időintervallumon belüli összesített M menetteljesítményét:

```
610 :  :  M$=RIGHT$("        "+STR$(M),7)
620 :  :  RR$=LEFT$(RR$,2)+"-"+RIGHT$(RR$,4)
630 :  :  PRINT#1," ";RR$;" : ";M$;" KM"
```

A karakteres kiírási formára itt azért van szükségünk, hogy a számadatok helyi érték szerint elhelyezkedve jobbrazárt számoszlopot képezzenek — feltéve, hogy a kilométerteljesítmények csak egész értékek lehetnek.
Mielőtt a program áttérne a következő gépkocsi, tehát az indextábla következő bejegyzésének feldolgozására, a kiírt teljesítményértéket az MM (mindösszesen) gyűjtőbe gyűjti:

```
640 :  :  MM=MM+M
699 :  NEXT I
```

A gépkocsik feldolgozásának befejeztével kiírja az összegsort, azaz a gépkocsik együttes MM menetteljesítményét:

```
710 :  PRINT#1
720 :  PRINT#1," ====================="
730 :  M$=RIGHT$("           "+STR$(MM),7)
740 :  PRINT#1," OSSZESEN:";M$;" KM"
```

Ezt követően megkérdezi, hogy akarunk-e további, a most megadottól esetleg eltérő intervallumon belül listát készíteni:

```
810 :  PRINT:PRINT
820 :  INPUT " VEGE =I/N= ";V$
830 :  IF V$= "I" THEN GOTO 910
840 :  IF V$<>"N" THEN GOTO 820
```

Ha akarunk, akkor a listakészítést az intervallum bekérésétől kezdve újraindítja:

```
850 :  PRINT "□"
860 :  PRINT:PRINT
899 GOTO 210
```

Ha úgy döntöttünk, hogy a listázásnak vége, akkor lezárja a kimenő állományt, a puffert és a parancscsatornát, majd leáll:

```
910 CLOSE 1
920 CLOSE 3
930 CLOSE 15
999 END
```

A rekordokat beolvasó szubrutin először betölti a blokkot a pufferba, egyben ellenőrzi a művelet sikerességét:

```
1010 PRINT#15,"B-R:"3;0;T;B
1020 GOSUB 9010
```

Most következik a rekord olvasása.

Ebben a programban azonban nem ellenőrizzük, hogy a rekordban tárolt rendszám azonos-e az indextáblában lévővel, ezért a rekordból a rendszámot be sem kell olvasnunk; egyszerűen átléphetjük, és a rekord olvasását közvetlenül a havi teljesítményadatokkal kezdhetjük. Ez azért oldható meg, mert a pufferhez tartozik egy mutató, angolul POINTER, amely a puffer olvasásakor mindig megmutatja, hogy a puffer hányadik bájtjáról kell beolvasni a soron következő adatot.

E mutatót (pointert) a lemezkezelő 0-ról indítva, a puffer kezelésével összhangban automatikusan állítja, de megfelelő lemezkezelő információk megadásával mi magunk is beállíthatjuk, illetve átállíthatjuk:

```
1030 PRINT#15,"B-P:"3;7
```

A mutatót beállító információkat a parancscsatornára adjuk ki. A műveletet meghatározó kulcsszó a BUFFER–POINTER. Általában a B–P rövidítést használjuk.

Ezután meg kell adnunk a puffer számára lefoglalt csatornát, ami esetünkben 3. Végül meg kell határoznunk azt a bájtpozíciót, amelyről a következő pufferkezelő műveletnek indulnia kell. Esetünkben a rendszám hat bájtját, valamint a szeparátorjelet akarjuk átléptetni, így az olvasásnak a 8. bájton, azaz a 7-es bájtpozíción kell elkezdődnie, mivel a bájtokat 0-tól számláljuk.

Megjegyezzük, hogy a puffer mutatója nemcsak olvasáskor, hanem íráskor is aktív, ez esetben mindig azt mutatja meg, hogy a puffer melyik bájtjáig van már feltöltve. Természetesen a mutató értékét ilyenkor is beállíthatjuk, szükség szerint 0 és 255 között bármilyen értékre. Ha nem élünk a mutató beállításának lehetőségével, mint ahogyan ezt a RANDOMKEZELES program lemezolvasó szubrutinjában tettük, akkor a lemezkezelő minden pufferfeltöltéskor automatikusan 0-ra állítja a mutatót, és a további pufferkezelő műveletekkel összhangban automatikusan kezeli.

Most végre következhet a rekord beolvasása, amely a fenti pufferbeállítás hatására a megadott bájtpozíciótól, vagyis a havi teljesítményadatoktól hajtódik végre:

```
1040 FOR J=1 TO 12
1050 :   INPUT#3,K(J)
1099 NEXT J
```

A lemezkezeléskor meghívott hibarutin hibajelzést ad, ha a lemezművelet sikertelen volt. Ilyenkor a rutin a nyitott csatornákat lezárja, majd leállítja a programot:

```
9010 INPUT#15,H,H$,TT,BB
9020 IF H<20 THEN GOTO 9999
9030 :   PRINT H;H$,TT;BB
9040 :   CLOSE 1
9050 :   CLOSE 3
9060 :   CLOSE 15
9099 :   STOP
9999 RETURN
```

A szubrutinban a RETURN utasítás egyrészt dokumentációs okokból szerepel, másrészt azért, hogy ne feledkezhessünk meg róla, ha a hibakezelést úgy írnánk át, hogy a program hiba esetén is folytatható legyen.

A programot gépeljük be, és mentsük ki RANDOMSOROS néven, de ne futtassuk le! A listázás elindítása előtt ugyanis célszerű a random állományba olyan adatokat felvinni, amelyekkel a listázás és a kigyűjtés helyessége könnyen ellenőrizhető. Töltsük be és indítsuk el ezért a RANDOMKEZELES programot. Vegyük sorra az AA 1111, FF 6666, DD 4444, BB 2222 rendszámmal azonosított rekordjainkat, és módosítsuk az adatokat úgy, hogy a havi teljesítmények az első gépkocsinál 100, a másodiknál 600, a harmadiknál 400, a negyediknél 200 kilométeresek legyenek minden egyes hónapban.
E módosítások végrehajtása után töltsük be és indítsuk el a RANDOMSOROS programot. Először kérjünk listát a januári teljesítményekről, azaz az 1. hónaptól az 1. hónapig.

Majd a második negyedévről, vagyis a 4. hónaptól a 6. hónapig:

```
♥♥ MENETTELJESITMENYEK ♥♥
--------------------------
 1. HONAPTOL   1. HONAPIG
--------------------------

AA-1111 :     100 KM
FF-6666 :     600 KM
DD-4444 :     400 KM
BB-2222 :     200 KM

====================
OSSZESEN:   1300 KM
```

```
♥♥ MENETTELJESITMENYEK ♥♥
--------------------------
 4. HONAPTOL   6. HONAPIG
--------------------------

AA-1111 :      300 KM
FF-6666 :     1800 KM
DD-4444 :     1200 KM
BB-2222 :      600 KM

====================
OSSZESEN:     3900 KM
```

Végül a teljes évről, tehát az 1. hónaptól a 12. hónapig:

```
♥♥ MENETTELJESITMENYEK ♥♥
--------------------------
 1. HONAPTOL  12. HONAPIG
--------------------------

AA-1111 :    1200 KM
FF-6666 :    7200 KM
DD-4444 :    4800 KM
BB-2222 :    2400 KM

====================
OSSZESEN:   15600 KM
```

Ennyi talán elég is ahhoz, hogy meggyőződjünk a program helyességéről. A listázást tehát leállíthatjuk.

AZ INDEXTÁBLA RENDEZÉSE

Az indextábla rendezésével elérhetjük, hogy a random állomány rekordjaira rendezett soros feldolgozást is végrehajthassunk.

Példánkban akár a csoportonkénti feldolgozásnak is lehetne értelme, mondjuk akkor, ha egy csoportba tartozónak tekintjük azokat a gépkocsikat, amelyeknek a rendszáma ugyanazzal a betűvel kezdődik.

A rendezés különösen akkor egyszerű, ha olyan kis indextáblánk van, hogy a tárban is rendezhető. A lehetséges rendezési módszerek közül egy olyat választottunk ki, amely egyszerű és biztonságos.

A rendezés hatásának könnyebb megfigyelése végett a rendezett indextáblát a létrehozása során nem visszük fel a lemezre, hanem a tárban tartjuk, amíg össze nem hasonlítjuk az eredetivel.

Megjegyezzük, hogy így a rendezés után két indextáblánk lesz: a rendezett és a rendezetlen. Ezt kizárólag a szemléltetés indokolja; a nem tanulás, hanem valódi felhasználás céljára készített programban ez felesleges.

Rendezési módszerünk az, hogy kiválasztjuk a rendezetlen indextábla legkisebb elemét, majd betöltjük a rendezett indextábla első helyére. Ezután megkeressük a rendezetlen táblában a második legkisebb rendszámot, és a neki megfelelő bejegyzés adatait átírjuk a rendezett indextábla második helyére. Így haladunk tovább mindaddig, amíg az új indextáblát teljesen fel nem töltjük.

Arról persze gondoskodnunk kell, hogy amikor a második legkisebb elemet keressük, akkor ne ismét az elsőt találjuk meg. Ezért valahányszor megtaláljuk a legkisebb elemet, az új indextáblába való áttöltése után legnagyobb elemet kell csinálnunk belőle, tehát a rendezetlen táblában át kell írnunk a rendszámot az összes többinél nagyobbra.

Ezzel az algoritmussal a bejegyzéseket rendszám szerint növekvő sorrendbe rendezzük. Ha csökkenő sorrendben kívánunk rendezni, ugyanígy járunk el, de a legkisebb elem helyett mindig a legnagyobb elemet kell választanunk.

A rendezőprogramunk tetszés szerint növekvő vagy csökkenő sorrendben tudja rendezni a 30 elemű indextáblát (16. ábra).

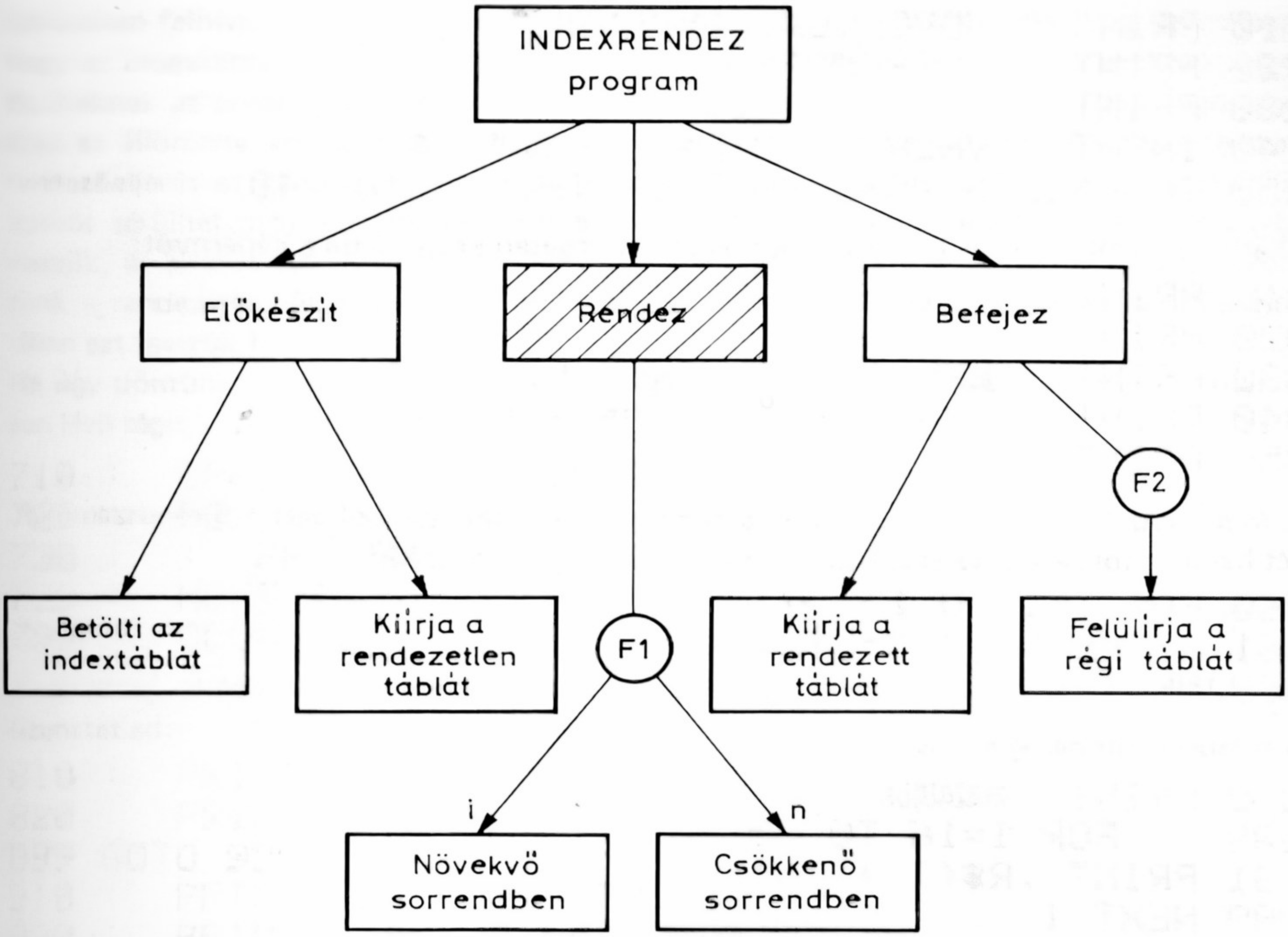
A feltételek:

- F1: ha növekvő sorrendet határoztunk meg, vagyis a V$ válasz = "N";
- F2: ha a régi indextábla felülírására engedélyt adtunk, azaz a "MEHET?" kérdésre a V$ válasz = "I".

A program maga az alábbiak szerint néz ki.

Először bejelentkezik:

```
1 PRINT "}"
2 PRINT
3 PRINT " INDEXTABLA RENDEZESE"
4 PRINT " ---------------------"
5 PRINT
6 PRINT
7 PRINT
```



16. ábra. Az indextábla rendezése

Majd definiálja az új RR$, TT, BB indextáblát, a régi R$, T, B indextáblát, a C$ legkisebb rendszámot, a P$ legnagyobb rendszámot és az S$ szeparátorjelet:

```
10 DIM RR$(30),TT(30),BB(30)
20 DIM  R$(30), T(30), B(30)
30 C$="******"
40 P$="ππππππ"
50 S$=CHR$(13)
```

Az indextábla üres helyeit jelölő csillagok alkalmasak a legkisebb rendszám szerepére, hiszen kódjuk minden betű kódjánál kisebb. Éppen ezért nem jelölhetik a legnagyobb rendszámot is, így növekvő sorrendű rendezés esetén a csillagokat ki kell cserélnünk a π jelre, mert az valóban nagyobb minden más karakternél.

A program betölti az indextáblát:

```
110 OPEN 2,8,2,"INDEXTABLA,SEQ,READ"
120 FOR I=1 TO 30
130 :   INPUT#2,R$(I),T(I),B(I)
139 NEXT I
199 CLOSE 2
```

Megkérdezi, hogy milyen irányú rendezést óhajtunk. Csak a növekvő (N) vagy csökkenő (C) választ fogadja el:

```
210 PRINT " NOVEKVO/CSOKKENO";
220 PRINT " SORRENDBEN?"
230 PRINT
240 INPUT " VALASZ =N/C=  N■■■";V$
250 IF V$<>"N" AND V$<>"C" THEN GOTO 240
```

Akármilyen sorrendet is határoztunk meg, függőlegesen kettéosztja a képernyőt:

```
310 PRINT "■"
320 PRINT
330 PRINT " EREDETI:",,"RENDEZETT:"
340 PRINT " ========",,"=========="
350 PRINT
```

A képernyő bal oldalán megjeleníti a rendezetlen indextáblából vett R$ rendszámokat, két hasábra tördelve. Az első hasábban van az első 15 rendszám:

```
360 FOR I=1 TO 15
361 :  PRINT "  "+R$(I)
369 NEXT I
```

A másodikban pedig a második 15:

```
370 PRINT "■■■■■■"
380 :  FOR I=16 TO 30
381 PRINT ,R$(I)+"  |"
389 NEXT I
```

Ezután a kérdésre adott válaszunk alapján növekvő vagy csökkenő sorrendben végrehajtja a rendezést, vagyis létrehozza az új indextáblát:

```
410 IF V$="N" THEN GOSUB 1010
420 IF V$="C" THEN GOSUB 2010
```

Ezután a képernyő jobb oldali felében megjeleníti a rendezett indextáblából vett RR$ rendszámokat, a rendezetlenhez hasonlóan 15—15 rendszámot tördelve egy-egy oszlopba:

```
510 PRINT "■■■■■■"
520 FOR I=1 TO 15
521 :  PRINT ,,RR$(I)
529 NEXT I
530 PRINT "■■■■■■"
540 FOR I=16 TO 30
541 :  PRINT ,,,RR$(I)
549 NEXT I
```

Ekkor választhatunk, hogy megtartjuk-e az eredeti, rendezetlen indextáblát, vagy megsemmisítjük, és mostantól fogva az újat, a rendezettet használjuk:

```
610 PRINT
620 INPUT " MEHET =I/N= ";V$
630 IF V$<>"I" THEN GOTO 910
```

Ismételten felhívjuk a figyelmet arra, hogy e lehetőség csak azért van a programunkban, hogy az indextáblát tetszésünk szerint akár többször is átrendezhessük, és ezeknek a rendezéseknek az eredményét kényelmesen tanulmányozhassuk. A program biztonságossága, azaz az állomány védelme ilyen módszert nem tesz szükségessé, hiszen egy jól tesztelt rendezőeljárás az állományt nem tudja elrontani, az csak a lemezreíráskor, illetve felülíráskor sérülhet meg. Ez pedig kivédhető, ha a rendezett indextáblát egy új állományba visszük, és sikeres létrehozása után töröljük a rendezetlent; vagy előbb másolatot készítünk a rendezetlenről, majd felülírjuk a rendezettel. (A felhasználói programokban általában ezt tesszük.)

Ha úgy döntünk, hogy jöhet az új, akkor a program az új indextáblával felülírja a lemezen lévő régit:

```
710 :   OPEN 2,8,2,"@:INDEXTABLA,SEQ,WRITE"
720 :   FOR I=1 TO 30
730 :   :   PRINT#2,RR$(I);S$;TT(I);S$;BB(I)
739 :   NEXT I
799 :   CLOSE 2
```

Akár az új, akár a régi indextábla mellett döntünk, a program döntésünknek megfelelő üzenetet ad:

```
810 :   PRINT "□":PRINT:PRINT
820 :   PRINT " MENT."
899 GOTO 999
910 :   PRINT "□":PRINT:PRINT
920 :   PRINT " AKKOR NEM MEGY!"
```

Végül leáll:

```
999 END
```

Most pedig lássuk magát a rendezési eljárást. Először növekvő sorrendben rendezünk, mert az a bonyolultabb.

A rendezőeljárás egy 30 menetes ciklusból áll, amelynek minden egyes menete rendre az új indextábla egy-egy elemét állítja elő:

```
1010 FOR I=1 TO 30
```

A program optimista módon feltételezi, hogy a rendezetlen táblának mindjárt az első eleme a legkisebb. Ennek megjegyzi az L indexét:

```
1020 :   L=1
```

Majd megvizsgálja a rendezetlen tábla összes többi R$ elemét, hogy kisebbek-e ennél a legkisebbnek tekintett elemnél:

```
1100 :   FOR J=2 TO 30
1110 :   :   IF R$(J)=C$ THEN R$(J)=P$
1120 :   :   IF R$(J)<R$(L) THEN L=J
1199 :   NEXT J
```

Persze, ha valamely bejegyzésben rendszám helyett C$ csillagokat talál, ki kell cserélnie őket a P$ π karakterre, különben mindig az üres helyek lennének a legkisebbek.

Ha valamely táblaelem nem kisebb a kiválasztott elemnél, akkor a program nem tesz mást, mint veszi a következő táblaelemet. Viszont ha valamely táblaelem kisebb a kiválasztottnál, akkor a kisebb táblaelem J indexét jegyzi meg, és a további hasonlításokban már ezt az elemet tekinti L kiválasztottnak, a legkisebbnek.

Amikor a rendezetlen tábla végére ér, a megjegyzett L index az adott menetben kiválasztható legkisebb R$ elem indexe lesz. Ekkor az általa meghatározott táblaelem adatait, az R$ rendszámot, a T sávcímet és a B szektorcímet átmásolja a rendezetlen táblából az új táblába:

```
1210  :   RR$(I)=R$(L)
1220  :   TT (I)=T (L)
1230  :   BB (I)=B (L)
```

Gondoskodik arról, hogy a következő menetben ne ugyanezt a táblaelemet találja legkisebbnek, azaz a rendszámot átírja a legnagyobb értékre, a P$ π karakterekre:

```
1240  :   R$ (L)=P$
```

Az új táblában, a rendezett indextáblában azonban nem lehetnek az üres helyeken π karakterek. A karbantartó program ugyanis csillagokat vár. Ezért ha az új táblába ilyen elem kerül, akkor ezen elemnél vissza kell állítani a C$ csillagokat:

```
1250  :   IF RR$(I)=P$ THEN RR$(I)=C$
```

A ciklus itt befejeződik:

```
1299 NEXT I
```

A szubrutin visszaadja a vezérlést a hívó programnak:

```
1999 RETURN
```

A csökkenő sorrend szerinti rendezés pontosan ilyen, de itt nem kell a csillagokat kicserélnünk és visszacserélnünk, továbbá természetesen nem a legkisebb, hanem a legnagyobb elemet kell az új táblába áttölteni, így a régi táblából is a legnagyobb elemet kell kiválasztanunk – ennek megfelelően a hasonlítás is fordítva történik:

```
2010 FOR I=1 TO 30
2020  :   L=1
2100  :   FOR J=2 TO 30
2110  :   :   IF R$(J)>R$(L) THEN L=J
2199  :   NEXT J
2210  :   RR$(I)=R$(L)
2220  :   TT (I)=T (L)
2230  :   BB (I)=B (L)
2240  :   R$ (L)=C$
2299 NEXT I
2999 RETURN
```

Figyeljünk fel arra, hogy a rendezőeljárás nem magát a legnagyobb (vagy legkisebb) R$ rendszámot jegyzi meg, hanem annak csak az L indexét. Így a táblába való betöltéskor a sávcímet és a szektorcímet nem kell keresnie a régi táblában, hanem közvetlenül elérheti őket a megjegyzett L index alapján.

Ennek további előnye, hogy a tábla rendezése nem a táblaelemek cserélgetésével valósul meg, ami lényegesen növelné a rendezés (amúgy is hosszú) idejét.

A begépelt rendezőprogramot mentsük ki a lemezre INDEXRENDEZ néven, majd bátran kipróbálhatjuk, sikertelen rendezés esetén sem romolhat el az indextáblánk.

A program elindítása után kérjünk csökkenő sorrend szerinti rendezést. A képernyőn összehasonlíthatjuk a rendezetlen és a rendezett indextáblát:

```
EREDETI:                    RENDEZETT
=========                   ============

AA1111      ******   |   FF6666       ******
FF6666      ******   |   DD4444       ******
DD4444      ******   |   BB2222       ******
BB2222      ******   |   AA1111       ******
******      ******   |   ******       ******
******      ******   |   ******       ******
******      ******   |   ******       ******
******      ******   |   ******       ******
******      ******   |   ******       ******
******      ******   |   ******       ******
******      ******   |   ******       ******
******      ******   |   ******       ******
******      ******   |   ******       ******
******      ******   |   ******       ******
******      ******   |   ******       ******

MEHET =I/N= ? N
```

A „MEHET?" kérdésre adjunk nemleges választ, majd indítsuk újra a programot úgy, hogy most már növekvő sorrendben rendezzen.

Ha a képernyőn hibátlanul megjelenik a rendezett állomány, a „MEHET?" kérdésre adjunk igenlő választ.

Javasoljuk, hogy ezután töltsük be és futtassuk le a RANDOMSOROS programot januártól decemberig tartó intervallumra:

```
** MENETTELJESITMENYEK **
-------------------------
 1. HONAPTOL  12. HONAPIG
-------------------------

AA-1111  :    1200 KM
BB-2222  :    2400 KM
DD-4444  :    4800 KM
FF-6666  :    7200 KM
=========================
OSSZESEN:    15600 KM
```

Így meggyőződhetünk arról, hogy noha a rendezőprogram a random állomány rekordjaihoz hozzá se nyúlt, a soros feldolgozóprogram mégis rendezett listát állít elő. Az indextáblás rendezésben éppen az a szép, hogy nem az állomány rekordjait rendezzük, hanem az indextáblát, amelynek bejegyzései lényegesen rövidebbek a rekordoknál.

Megjegyezzük, hogy nem az itt bemutatott rendezési eljárás az egyetlen, és nem is feltétlenül ez az optimális megoldás. Kiválasztásában nem gyakorlati, hanem didaktikai szempontok vezettek.

A rendezett indextábla egyik legelőnyösebb tulajdonsága az, hogy benne nemcsak lineárisan lehet keresni, mint a rendezetlen állományokban, hanem a rendezett állományokra jellemző keresési eljárások is használhatók. (Ezekkel a „SOROS LEMEZÁLLOMÁNYOK'' című kötetben foglalkoztunk.) Ha a rendezett indextábla a tárban van, optimálisabb keresési algoritmusokat is alkalmazhatunk. Ezekkel a random állományok rekordjainak közvetlen elérése rendkívül meggyorsítható; hiszen minden egyes rekord elérése előtt szükség van az indextáblában a rekord címének megkeresésére. Ilyenkor persze gondosan ügyelnünk kell arra, hogy a tábla rendezettsége a feldolgozás, például a karbantartás közben ne romolhasson el. Ez természetesen bizonyos mértékig kényelmetlenebbé teszi a tábla kezelését. Ha választási lehetőségünk van, minden alkalommal meg kell fontolnunk, hogy a keresés meggyorsulásából származó előny elegendő-e a kezelés bonyolultságának kiegyenlítésére.

Felhívjuk a figyelmet arra, hogy a példában használt megoldás az úgynevezett teljes indexelés. Ez azt jelenti, hogy minden rekordhoz tartozik bejegyzés az indextáblában. Készíthető és használható azonban olyan indextábla is, amely nem minden rekord azonosítóját és címét tartalmazza, hanem mondjuk csak minden tizedikét. A közbülső rekordok címét ilyenkor a táblában szereplő szomszédos rekordok címéből kell előállítani. Ehhez persze megfelelő kulcseloszlás, rendezettség és címképzési eljárás (például interpoláció) szükséges.

Végezetül érdemes még megemlíteni, hogy a bemutatott program úgynevezett egyszintű indexeléssel dolgozik, vagyis az indextáblában lévő cím közvetlenül a rekordra mutat.

Az indextáblák hátránya, hogy általában akkor kezelhetők kényelmesen, ha a tárban vannak — a tár azonban véges; nagyobb állományok indextáblái nem férnek el benne.

Ilyenkor az egyik sikeres megoldás az indextábla feldarabolása lehet. A gépkocsik esetében például megtehetjük, hogy nem egy, hanem 26 indextáblát hozunk létre, külön egyet az A-val, egyet a B-vel, és így tovább, egyet a Z-vel kezdődő rendszámú gépkocsik részére. Ezek közül csak azt az egyet tartjuk a tárban, amelyikre az adott rendszámú gépkocsi címének megkereséséhez szükségünk van. Minthogy a megfelelő indextábla használat előtti betöltése, majd használat utáni kimentése időigényes, ez a megoldás akkor gazdaságos, ha egy-egy indextáblát hosszabb ideig használhatunk, például ha a gépkocsik feldolgozása rendszám szerint rendezett.

Egy másik megoldás lehet az úgynevezett többszintű indexelés, amikor is két indextáblánk van: az egyik a rekordazonosítókat és a hozzájuk tartozó címeket tartalmazza, mégpedig rendezetten; a másik, általában jóval kisebb indextábla pedig azt tárolja, hogy az egyes rekordcsoportok a „nagy'' indextábla mely részében találhatók.

A gépkocsik esetén ez utóbbi indextábla például 26 elemű lehet, ahol az első elem meghatározza, hogy az A betűvel kezdődő rendszámú gépkocsik bejegyzései hányadik elemtől hányadik elemig töltik ki a „nagy'' indextáblát; a második elem ugyanilyen információt tartalmaz a B-s rendszámú gépkocsikról; és így tovább, egészen Z-ig. Ekkor állandóan

bent tartjuk a tárban a „kis" indextáblát, valamint a „nagy"-nak mindig csak azt a részét, amelyet az előbbi a rekordazonosító szerint meghatároz. A betöltésből, kimentésből származó időveszteséggel természetesen itt is számolnunk kell. Emellett bonyolítja a helyzetet, hogy a két indextáblát mindig egymással összhangban kell módosítanunk. (A bonyolultabb indextáblákkal a relatív állományok tárgyalásakor még foglalkozunk.)

A RANDOM ÁLLOMÁNYOK KEZELÉSÉNEK ÖSSZEFOGLALÁSA

A random szervezési mód alapja az, hogy a lemezen lévő blokkok mindegyike egyértelműen azonosítható annak a sávnak és szektornak a címével, amelyen elhelyezkedik.
Amit random állománynak nevezünk, valójában nem más, mint a lemezen elszórt blokkok pusztán logikailag összetartozó együttese.
A random szervezésű állomány legfeljebb 664, egyenként 256 bájt méretű blokkból állhat. Ezen blokkok egymástól függetlenül és közvetlenül elérhetők, de a tartalmukhoz – akár felíráskor, akár beolvasáskor – csak pufferen keresztül lehet hozzáférni. Maga az állomány azonban, mint egység, nem kezelhető.
A random állományt sem létrehozni, sem törölni, sem megnyitni, sem lezárni nem lehet. Neve nincs, és a lemez tartalomjegyzékében sem szerepel. Így random állománykezelő parancsok sem léteznek. A blokkokat és a puffert azonban külön lemezparancsokkal, illetve utasításokkal kell kezelni.
Egy lemezen akárhány random állomány lehet, akár úgy is, hogy egy-egy blokk egyidejűleg több random állománynak is eleme.
A random állomány kezeléséhez mindig két csatornát kell nyitva tartanunk: a parancscsatornát és egy adatcsatornát, a blokk-kezelő parancsok, illetve a puffer számára.
Az alábbiakban összefoglaljuk a random állomány kezelésével kapcsolatos utasításokat.

OPEN: a csatornák megnyitása

```
OPEN állomány,egység,parancscsatorna
OPEN állomány,egység,adatcsatorna,"#puffer"
```

Hatása: megnyitja a parancscsatornát, majd a megadott puffer számára kijelölt adatcsatornát is. Például:

```
OPEN 15,8,15
OPEN 3,8,3,"#2"
OPEN 4,8,4,"#"
```

A puffert 0-tól 2-ig terjedő sorszámmal azonosíthatjuk. Ha nem adunk meg sorszámot, akkor a puffert a lemezkezelő rendszer automatikusan jelöli ki.

BLOCK–ALLOCATE: blokk lefoglalása az állomány számára

```
PRINT#15,"B-A:"0;sáv;szektor
```

Hatása: a lemeztérkép alapján megállapítja, hogy a megadott címen lévő blokk szabad-e, avagy már foglalt. Ha szabad volt, lefoglalja és feljegyzi a lemeztérképre. Például:

```
PRINT#15,"B-A:"0;21;13
```

A blokk állapotát a parancscsatornáról olvashatjuk le.

INPUT: a parancscsatorna olvasása

INPUT#15,hibakód,hibaüzenet,sáv,szektor

Hatása: ha a blokk szabad volt, akkor **0** hibakódot, **"OK"** hibaüzenetet, **0** sávcímet és **0** szektorcímet kapunk. Ha viszont foglalt a blokk, akkor a hibakód **65**, a hibaüzenet **"NO BLOCK"** lesz, a sávcím és a szektorcím pedig a legközelebbi magasabb című szabad blokk helyét adja meg. Sikertelen művelet (hibás, rosszul behelyezett, megtelt lemez stb.) esetén a hibának megfelelő kódot és üzenetet kapunk. Például:

INPUT#15,H,H$,SA,SZ

A parancscsatorna csak BLOCK–ALLOCATE parancs után szolgáltatja ezeket az értékeket; más esetben a már ismert, szokásos módon működik.

BLOCK–FREE: lefoglalt blokk törlése

PRINT#15,"B–F:"0;sáv;szektor

Hatása: törli a blokk foglaltságát jelző bitet a lemeztérképről, de magát a blokkot nem. Ezzel azonban a blokk már felszabadul, nem tartozik többé a random állományhoz. Például:

PRINT#15,"B–F:"0;TR;BL

Téves sáv- és szektorcím megadása esetén a random állományhoz nem tartozó blokkot, azaz más állomány blokkját is törli, ha az foglalt volt.

PRINT: a puffer feltöltése adatokkal

PRINT#állomány,adatok

A puffer tetszőlegesen tölthető fel adatokkal a soros állományoknál szokásos PRINT utasítással. Például:

PRINT#3,A$;S$;A%;S$;A

A lemezre mindig a puffer aktuális tartalma kerül. Ezért a lemezre írandó adatokat a lemezrevitel előtt mindig a pufferba kell betölteni, abban a formában, amilyenben a lemezen lévő blokkban tárolandók.

BLOCK–WRITE: blokk felírása a lemezre

PRINT#15,"B–W:"csatorna;0;sáv;szektor

Hatása: az adott csatornához rendelt puffer aktuális tartalmát felírja a lemez megadott sávjának megadott szektorára. Például:

PRINT#15,"B–W:"3;0;SA;SZ

Ha a megadott cím létezik, mindig felírja a blokkot, akár szabad a helye, akár már lefoglalta más állomány (például a lemez tartalomjegyzéke).

BLOCK–READ: blokk betöltése a lemezről

PRINT#15,"B–R:"csatorna;0;sáv;szektor

Hatása: az adott címen lévő blokkot betölti a megadott csatornához rendelt pufferbe, ha a blokk nem szabad. Például:

PRINT#15,"B–R:"3;0;16;12

A foglalt blokkot akkor is betölti, ha nem a random állományhoz tartozik.

INPUT: adatok olvasása a pufferből

INPUT#állomány, változók

A pufferba betöltött adatok a soros állományoknál szokásos INPUT utasítással olvashatók ki a pufferből. Például:

INPUT#3,A$,A%,A

GET: bájt olvasása a pufferből

GET#állomány,karakteres változók

A pufferba betöltött bájtok a soros állományoknál szokásos GET utasítással olvashatók ki a pufferből. Például:

GET#3,B$

BUFFER–POINTER: bájtpozíció beállítása

PRINT#15,"B–P:"csatorna;pozíció

Hatása: a soron következő pufferba írás, illetve pufferből olvasás az adott csatornához rendelt puffer megadott bájtpozíciójánál fog elkezdődni. Például:

PRINT#15,"B–P:"3;123

A bájtpozíció 0 és 255 között állítható. Segítségével egy blokkba több rekord is felvihető, illetve a blokk adatai közvetlenül is elérhetők. Ha a pozíciót nem állítjuk be, értéke mindig az utolsó beírt vagy kiolvasott bájtot követő bájtpozíció; új blokk esetén ennek megfelelően mindig 0.

CLOSE: a csatornák lezárása

CLOSE állomány

Használat után a csatornákat a szokásos módon le kell zárni.

Először mindig a puffer csatornáját, majd a parancscsatornát. Például:

CLOSE 3
CLOSE 15

A random állomány kezelésének azonban sajátos logikája van. Nem maguk a fenti utasítások a nehezek, hanem az, hogy közülük mikor, melyiket, milyen sorrendben és milyen paraméterekkel használjuk. Mindezekről a mintaprogramoknál már volt szó, de itt is összefoglaljuk a random állománykezelés sajátosságait.

Alapvetően a három fő funkcióból kell kiindulnunk: a blokk felviteléből, beolvasásából és törléséből.

— **FELVITEL** esetén

BUFFER–POINTER: a puffer mutatójának beállításával meghatározhatjuk a puffer azon bájtját, amelytől a feltöltése elkezdődhet.

PRINT: tetszőleges adatokkal feltölthetjük a puffert vagy a mutató beállításával kiválasztott részét.

BLOCK–ALLOCATE: lefoglalhatjuk a lemez egy blokkját; ennek sávcímét és szektorcímét tetszőlegesen határozhatjuk meg.

INPUT: a parancscsatornáról leolvashatjuk, hogy a blokk lefoglalása sikeres volt-e, avagy sem; és ha nem, megkaphatjuk a legközelebbi szabad blokk címét.

BLOCK–WRITE: a lefoglalt blokkba felvihetjük a puffer aktuális tartalmát.

INPUT: a parancscsatorna lekérdezésével meggyőződhetünk a blokk felírásának sikerességéről.

— **BEOLVASÁS** esetén

BLOCK–READ: a tetszőlegesen megadott sávcímen és szektorcímen elhelyezkedő blokk tartalmát betölthetjük a pufferba.

INPUT: a parancscsatorna lekérdezésével meggyőződhetünk a lemezolvasás sikerességéről.

BUFFER–POINTER: a puffer mutatójának beállításával meghatározhatjuk a puffer azon bájtját, amelytől az olvasás elkezdődhet.

INPUT, GET: a puffer tartalmát vagy a mutató beállításával kiválasztott részét adatonként vagy bájtonként olvashatjuk.

— **TÖRLÉS** esetén

BLOCK–FREE: a lefoglalt blokk felszabadításával törölhetjük a random állomány tetszőleges blokkját.

VALIDATE: a lemez tömörítésével (újraszervezésével) törölhetjük a random állomány (egyben a lemezen lévő minden random állomány) összes blokkját.

A parancscsatorna figyelésével kapcsolatban megjegyezzük, hogy blokk felírása, azaz **BLOCK–WRITE** előtt feltétlenül hajtsunk végre **BLOCK–ALLOCATE** parancsot, majd olvassuk be a parancscsatornáról a hibakódot, és ennek értékétől tegyük függővé az írás végrehajtását.

A parancscsatornán elérhető hibakód és hibaüzenet vizsgálata más lemezműveletek után is célszerű lehet, minthogy hiba bárhol előfordulhat. Ezek hatása azonban legtöbbször „csak" annyi, hogy a program hibás eredményt állít elő vagy futása megszakad. A hibavizsgálat igazán a blokk felírásakor fontos, hiszen ez az a művelet, amely igazi katasztrófát tud okozni azzal, hogy felülírhatja a lemezen lévő bármely állományunkat, programunkat, sőt magát a lemez nyilvántartását (a lemeztérképet vagy a tartalomjegyzéket) is; így akár a teljes lemezt használhatatlanná tehetjük.

Végül következzék néhány megjegyzés az indextábla, illetve a random állományok használatához.
Az indextáblának valójában semmi köze a random állományhoz, amely nélküle is kezelhető. Az indextábla csupán egy hasznos segédeszköz, mely lehetővé teszi annak az egyértelmű számontartását, hogy melyik rekord melyik blokkon van, és megfordítva, hogy melyik blokk melyik rekordot tartalmazza; így a blokk kiválasztását teljesen a gépre bízhatjuk, azaz minden rekordot a lemezkezelő „keze ügyébe eső" első szabad blokkra írhatunk.

Indextábla nélkül a kulcskérdés a rekord és a blokk egymáshoz rendelése. Ha sikerül megoldanunk, hogy a rekordokhoz később megismételhető módon ki tudjunk választani egy blokkot, akkor a rekordot erre a blokkra bátran felvihetjük, mert később ugyanezzel a kiválasztással meg is fogjuk találni.

Egyébként az indextábla használata főleg akkor kényelmes, ha elfér a tárban. Ilyenkor ugyanis a karbantartása egyszerű, és közvetlen eléréssel megvalósítható.
Ha az indextáblát a mérete miatt vagy más okból nem tudjuk a tárba tölteni, akkor feldolgozás közben is a lemezen kell tartanunk. Ilyenkor általában az indextáblának csak egy részét töltjük be a tárba, és feldolgozás közben ezt a betöltött részt szükség szerint cserélgetjük. Ez az indextábla karbantartását, de még a keresését is jelentősen lelassítja és körülményessé teszi. Ezért erősen meg kell fontolnunk, hogy ilyenkor nem érdemesebb-e egy másik állományszervezési módot választani.
Megjegyezzük, hogy a példában bemutatott indextábla a lehető legegyszerűbb, mondhatni a legprimitívebb. Ennél sokkal rafináltabb táblák is használhatók, amelyek gyorsabb keresést tesznek lehetővé. A rendezett táblán kívül példaként csak egy érdekes táblát említünk meg: az úgynevezett önátrendező táblát, amelynek mindig az elején helyezkednek el a leggyakrabban használt bejegyzések.
Felhívjuk a figyelmet arra, hogy a random állományokkal úgynevezett listaszerkezetek is megvalósíthatók, tehát ez az állományszervezési mód messze túlmutat az egyszerű adatfeldolgozás keretein.*

A random állományok lehetőségei más tekintetben is nagyobbak a mintapéldákban bemutatottnál. Egyrészt abból, hogy a példákban egy rekordot egy blokkra vittünk, nem következik az, hogy egy blokkra nem vihető fel több rekord. Ha gondoskodunk arról, hogy a blokkon lévő rekordok egymástól megkülönböztethetők legyenek, és megoldjuk, hogy egyik rekorddal se tudjuk véletlenül felülírni a blokkon már fent lévő másikat, akkor nincs elvi akadálya, hogy egy blokk logikailag több rekordból álljon. Márpedig mindez, ha körülményesen is, de megvalósítható.
A legegyszerűbb esetben, ha minden rekordunk azonos hosszúságú, a megoldás nem is bonyolult, hiszen a puffer mutatójának (BUFFER–POINTER) beállításával könnyen elérhetjük, hogy minden rekord a neki megfelelő pozícióra kerüljön. A dolog azért mégsem ilyen egyszerű. Ugyanis egy blokk egyidejűleg tartalmazhat feltöltött, üres és törölt rekordot is, így a blokk és a rekord állapota nincs összhangban. Ezért amikor egy rekord

*Akit bővebben érdekelnek a leképezési eljárások, a táblák és listaszerkezetek kezelésének lehetőségei, az utánanézhet Mérey András „Adatszerkezetek" című könyvében (SZÁMOK, Budapest, 1979), illetve Bodor Tibor „Adatszerkezetek I–IV." című oktatófilm-sorozatában (SZÁMOK, Budapest, 1981).

számára szabad helyet keresünk, nem hagyatkozhatunk kizárólag a BLOCK–ALLOCATE parancsra, hanem a foglalt blokkban is meg kell tudnunk találni az üres vagy törölt rekordhelyeket.

Megjegyezzük, hogy a puffer mutatóját nemcsak a rekord elejére állíthatjuk be, hanem akár minden egyes adat elejére is; feltéve persze, hogy tudjuk az adat pozícióját. Ilyenkor a blokk minden egyes adata, sőt GET utasítással való olvasás esetén minden egyes bájtja közvetlenül elérhető, mégpedig tetszőleges sorrendben és akárhányszor, hiszen a mutató visszafelé is állítható.

Ugyancsak nem következik a példákból, hogy a rekord mérete nem haladhatja meg a puffer, illetve a blokk méretét. Természetesen több adat nem írható egy blokkra, mint amennyi elfér benne, de semmi akadálya sincs annak, hogy a rekordunk egy része egy bizonyos blokkon, a másik része pedig egy másik blokkon legyen. Ez csak szervezés kérdése, tehát megoldható, noha ugyancsak nem egyszerűen.

Az sem igaz, hogy egy lemezoldalon csak egyetlen random állomány lehet. Valójában akárhány állományt létrehozhatunk, hiszen az amúgy is csak egy logikai egység. Más kérdés, hogy miként tartjuk számon, hogy melyik blokk melyik állományhoz tartozik, és miként oldjuk meg, hogy az egyik állomány kezelésekor ne sértsük meg a másik blokkjait. Ugyanis egy blokk egyidejűleg akár több random állományhoz is tartozhat. Ez csak leképezés kérdése. A lemezkezelő számára egyetlen random állomány sem létezik: csak foglalt és szabad blokkokat ismer, más különbséget nem tesz blokk és blokk között, ha nem tartozik bejegyzett állományhoz, programhoz vagy a lemeznyilvántartáshoz.

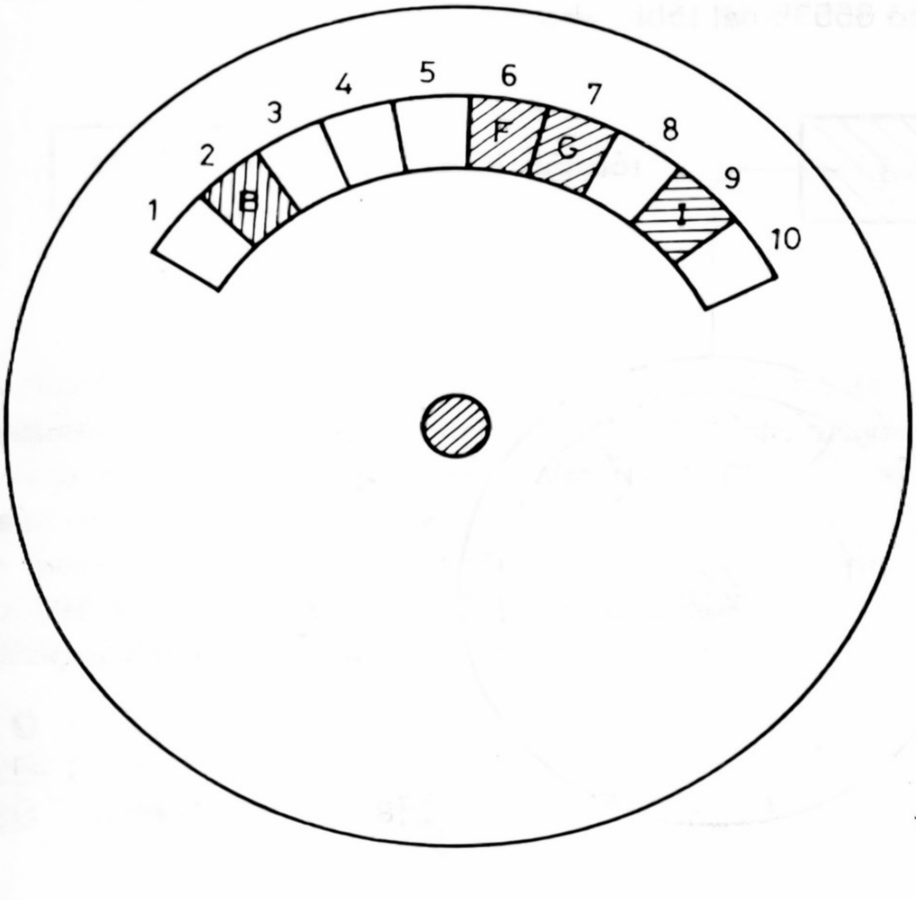
II. RÉSZ
RELATÍV ÁLLOMÁNYOK

A relatív állományok kezelése

A relatív állományszervezési mód arra az alapelvre épül, hogy egy azonos méretű elemekből álló tárolóegységen belül minden egyes elem helye egyértelműen meghatározható a kérdéses elem sorszámával. Ezt használja ki a tárkezelő, amikor megengedi, hogy a tárban egydimenziós tömböket, vektorokat hozzunk létre.

Meg kell adnunk a tömb méretét, például a DIM T(8) utasítással. Amikor egy tömbelemre a sorszámával hivatkozunk (például az A = T(3) utasításban), akkor a tárkezelő ahelyett, hogy a tárban keresné, a tömb kezdőcíméből, az elem sorszámából és az elemek hosszából kiszámítja az adott tömbelem címét. Így a tömb elemei közvetlenül elérhetők.

Hasonló adattárolási módot tesz lehetővé a lemezen a COMMODORE lemezkezelője (17. ábra):



17. ábra. Relatív állomány

Itt egy alkalmasan előkészített lemezterületet kell létrehozni, amely azonos méretű rekordokra oszlik. Ez az úgynevezett relatív állomány. Ennek létrehozásához természetesen meg kell adnunk a rekordok számát és méretét. A rekordjaink közvetlen eléréséhez a lemezkezelő ugyanazt az alapelvet használja ki, mint amelyre a tárkezelő is támaszkodott a tömbelemek elérésekor.

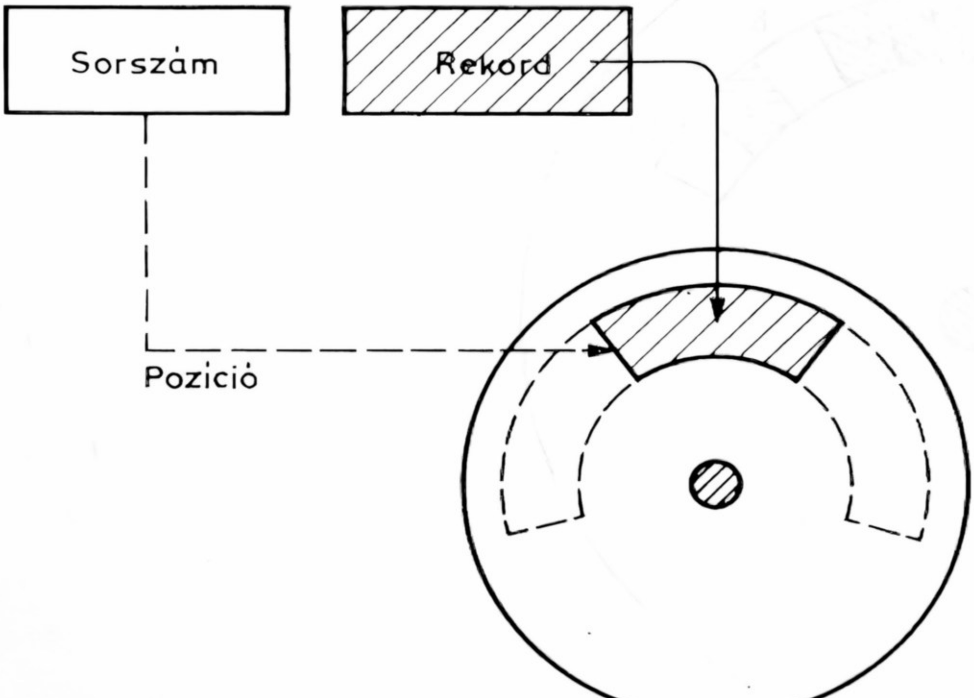
FIGYELEM!
Csak az alapelv azonos, a végrehajtás módja egy kicsit bonyolultabb. Ugyanis a rekordok által elfoglalt lemezterület nem feltétlenül folytonos. Könnyen belátható, hogy már az állomány létrehozásakor sem mindig biztosítható, hogy a megfelelő méretű lemezterület összefüggő legyen; de ha ez sikerül is, az végképp nem garantálható, hogy az állomány bővülésekor a lemezterület folytonossága megmaradjon. A lemezkezelőt felkészítették arra, hogy az állomány különböző helyen tárolt szektorait és ezek sorrendjét nyilvántartsa, így a relatív állomány számunkra olyannak látszik, és úgy is kezelhető, mintha a rekordok folyamatosan követnék egymást.

Ha megadjuk egy rekord sorszámát, akkor a lemezkezelő, minthogy tudja az állomány kezdőcímét, valamint a rekordok hosszát, egyszerűen kiszámítja a rekord fizikai helyét a lemezen. Felíráskor ide viszi fel a rekordot, beolvasáskor pedig innen veszi elő (18. ábra).

A programozónak nem is kell tudnia, sőt ténylegesen nem is ismeri a rekord abszolút helyét. Hivatkozáskor csupán a rekordsorszámot kell megadnia, tehát a rekordnak csak az állomány kezdetéhez viszonyított pozícióját. (Innen származik a relatív elnevezés.) A többi már a lemezkezelő dolga.

A lemez tárolási módja, valamint a lemezkezelő működési sajátosságai miatt tekintettel kell lennünk néhány megszorításra:

— A relatív állomány mérete nem haladhatja meg a 167132 bájtot.
— A rekordok hossza legfeljebb 254 bájt lehet.
— Az állományban nem tárolható 65535-nél több rekord.



18. ábra. Relatív elérés

- Az állomány lemezterülete legfeljebb 720 blokkot foglalhat el. Ezt a korlátot a kislemezen természetesen nem tudjuk megsérteni, hiszen legfeljebb 664 szabad blokk van rajta.

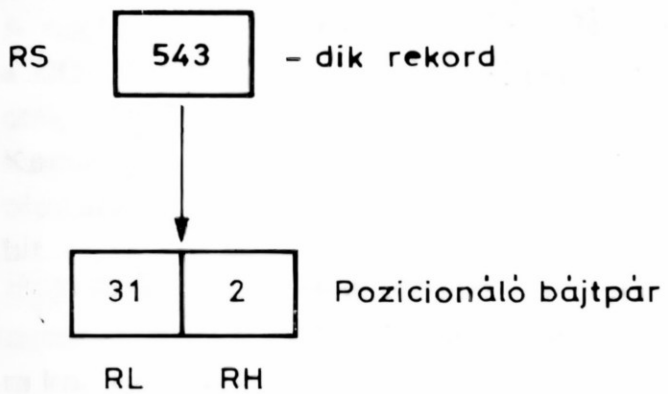
Ezzel szemben:

- A relatív állomány rekordjaiból annyi helyezkedhet el egy blokkon, amennyi elfér. Így a lemezterület kihasználtsága igen gazdaságos.
- A relatív állománynak nem kell a lemezen összefüggő területet elfoglalnia, mert a lemezkezelő külön nyilvántartást vezet a relatív állományhoz tartozó blokkokról.
- Az állomány számára kijelölt lemezterületet a lemezkezelő a fenti korlátokon belül szükség szerint automatikusan bővíti.
- Megfelelő pozicionálással nemcsak a rekordok, hanem az adataik, sőt a bájtjaik is közvetlenül elérhetők.

Az első két lehetőséget a lemezkezelő a programozótól függetlenül biztosítja. A második kettő kihasználása azonban már a programozón is múlik.

A REKORDSORSZÁM MEGHATÁROZÁSA

Ha egy rekordra hivatkozni akarunk, meg kell határoznunk, hogy az állomány elejétől számított hányadik rekord, vagyis meg kell adnunk a sorszámát. Ezt a műveletet a COMMODORE szóhasználatával pozicionálásnak is nevezzük. Végrehajtására külön utasítás van; ezzel majd egy példa kapcsán később fogunk megismerkedni.
Most csak azzal foglalkozunk, hogy milyen formában kell a rekord sorszámát megadnunk. Ez ugyanis a szokásostól eltérő (lásd 19. ábra).



19. ábra. A pozíció kiszámítása

A rekordsorszámot két pozicionáló bájtra kell felbontani, és a pozicionáláskor a sorszám decimális alakja helyett ezeket kell használnunk, mégpedig úgy, hogy először az alacsonyabb helyi értékű, úgynevezett alsó (low), majd a magasabb helyi értékű, úgynevezett felső (high) bájtot adjuk meg.
A pozicionáló bájtok kiszámítása a rekordsorszámból egyszerű művelettel történik: az RH felső bájt értéke az RS rekordsorszám és a 256 konstans hányadosának egész része; az RL alsó bájt pedig a fenti osztás maradéka:

```
10 PRINT "J"
20 PRINT
30 INPUT " REKORDSORSZAM=";RS
```

```
40 IF RS<0 OR RS>65535 THEN GOTO 10
50 IF RS=0             THEN GOTO 99
60 RS=INT(RS)
61 RH=INT(RS/256)
62 RL=RS-RH*256
70 PRINT "]";"
71 PRINT "]";" RS=";RS;"    ",
72 PRINT      "RL=";RL,
73 PRINT      "RH=";RH
80 GOTO 20
99 END
```

Futtassuk le a programot, rendre **111, 222, 333, 444, 555, 666, 777, 888, 999, 65535** és végül **0** sorszámot megadva:

```
RS= 111
RS= 222
RS= 333
RS= 444
RS= 555
RS= 666
RS= 777
RS= 888
RS= 999
RS= 65535
REKORDSORSZAM=? 0
```

A művelet fordítva is elvégezhető: ha az RH felső bájtot megszorozzuk 256-tal, és a szorzathoz hozzáadjuk az RL alsó bájtot, akkor az RS rekordsorszámot kapjuk:

```
10 PRINT "]"
20 PRINT
30 INPUT " ALSO & FELSO BAJT=";RL,RH
40 RL=INT(RL)
41 RH=INT(RH)
50 IF RL<0 OR  RL>255 THEN GOTO 10
51 IF RH<0 OR  RH>255 THEN GOTO 10
52 IF RL=0 AND RH=0   THEN GOTO 99
60 RS=RH*256+RL
70 PRINT "]";
71 PRINT "
72 PRINT "] RL=";RL,"RH=";RH,
73 PRINT "         RS=";RS
80 GOTO 20
99 END
```

Ezt is futtassuk le, az alsó és felső bájtoknak rendre (1,0); (0,1); (1,1); (2,1); (1,2); (2,2); (255,255); végül (0,0) értékpárt adva:

```
RL= 1      RH= 0
RL= 0      RH= 1
RL= 1      RH= 1
RL= 2      RH= 1
RL= 1      RH= 2
RL= 2      RH= 2
RL= 255    RH= 255
ALSO & FELSO BAJT=? 0,0
```

E mintaprogramok az irreális, azaz a negatív vagy a meg nem engedett magas rekordsorszámokat, illetve az alsó és felső bájtokat nem fogadják el, így mindig korrekt eredményt adnak. Más hibavédelem azonban nincs beépítve, ezért ügyeljünk az adatok helyes begépelésére; ellenkező esetben a kiírás nem lesz áttekinthető. Különösen a második programnál ne feledkezzünk meg arról, hogy az alsó és felső bájtot egyszerre kell megadnunk, ebben a sorrendben és egymástól vesszővel elválasztva.

Megjegyezzük, hogy az RH és RL pozicionáló bájtok tulajdonképpen 256-os számrendszerben adják meg az RS decimális rekordsorszámot. Erre azért van szükség, mert a lemezkezelő a rekordpozíciót fixpontos bináris formában kéri. Ez pedig a 256-os számrendszerben felírt értékből CHR$(RL)+CHR$(RH) alakban egyszerűen előállítható.

A decimálisról binárisra való konvertálásról azért kell magunknak gondoskodnunk, mert a COMMODORE ugyan minden numerikus értéket automatikusan binárisan ábrázol, de csak 32767-ig fixpontosan; az ennél nagyobb értékeket már lebegőpontosan.

Konverziónkkal a 65535-öt is fixpontosan tudjuk ábrázolni. Az eltérés oka az, hogy a gépi számábrázolás 1 bitet az előjelnek tart fenn, így a számjegyek ábrázolására csak 15 bit marad. Konverziónk azonban a 16. bitet is értékes számjegyként használja fel. (Így negatív számokat nem tudunk ábrázolni, de amúgy is csak a pozitív rekordsorszámoknak van értelmük. Ugyanakkor nem kell a relatív állomány méretét 32767 rekordra korlátoznunk.)

A BÁJTPOZÍCIÓ BEÁLLÍTÁSA

A rekordpozíción kívül mind íráskor, mind olvasáskor meg kell adnunk a rekordon belül azt a bájtot, amelyen az adatok írása, illetve az adatok olvasása elkezdődhet. Így a rekord tetszőleges számú bájtja átléptethető.

FIGYELEM!

A rekordon belüli bájtok számlálása az általános szokástól eltérően nem 0-tól, hanem 1-től kezdődik, és elvileg 255-ig tart. Minthogy a relatív állomány rekordjainak hossza nem haladhatja meg a 254 bájtot, a 255-ös bájtpozíciót nem használhatjuk ki, sőt — mint később látni fogjuk — értékes adatunk legfeljebb a 253-as bájtra kerülhet még.

Természetesen arra ügyelnünk kell, hogy a megadott bájtpozíció ne mutasson ki a tényleges rekordterületről; vagyis például egy 132 bájtos rekordban nem pozicionálhatunk a 234. bájtra, noha ez formailag megengedhető lenne.

AUTOMATIKUS BŐVÍTÉS

Említettük, hogy a relatív állomány csak meghatározott számú rekordból álló, előkészített lemezterületen hozható létre. Előfordulhat azonban, hogy ez a terület kicsinek bizonyul, vagy azért, mert rosszul becsültük meg az állomány méretét, vagy azért, mert (önhibánkon kívül) ,,kinőtte'' a lemezterületet.

Ez azonban nem korlátozza a relatív állomány használhatóságát. A lemezkezelő ugyanis a lemezterületet szükség szerint kiterjeszti, ha az állományba olyan rekordot akarunk felvinni, amely már nem fér el oda, vagyis amelynek sorszáma nagyobb, mint a létező állomány utolsó rekordjának a sorszáma (20. ábra).

A kiterjesztés azt jelenti, hogy ha például egy állományunkban 10 rekord van, és mi mondjuk egy 15-ös sorszámú rekordot kívánunk felvinni, akkor a lemezkezelő létrehozza a 11-es, 12-es, 13-as és 14-es rekordokat, majd felírja a 15-ös rekordot. Így az állomány mérete automatikusan megnövekszik.

Megjegyezzük, hogy az a blokk, amelyen a relatív állománynak akár csak egyetlen rekordja is van, más állományokhoz már nem használható fel. Ezért a lemezkezelő mind létrehozáskor, mind bővítéskor teljesen feltölti üres rekordokkal a megkezdett blokkot akkor is, ha így a rekordok száma nagyobb lesz az általunk megadottnál.

Most pedig lássuk a relatív állomány lemezterületének már eddig többször is említett, de még soha meg nem magyarázott előkészítését!

A RELATÍV ÁLLOMÁNY LÉTREHOZÁSA

Az előkészítés lényege, hogy a lemezkezelő annyi üres rekordot hoz létre, amennyit meghatározunk, és ezeket felviszi a lemezre. Ehhez persze a rekordok hosszát is meg kell adnunk (21. ábra).

Megjegyezzük, hogy — mint már említettük — a megkezdett blokkok mindenképpen feltöltődnek, tehát a példánkban bemutatott állományba akkor is 9 rekord kerül, ha csak 7 létrehozására adunk utasítást. Látható továbbá, hogy a szektor-, illetve sávhatárok a rekordokat nem befolyásolják.

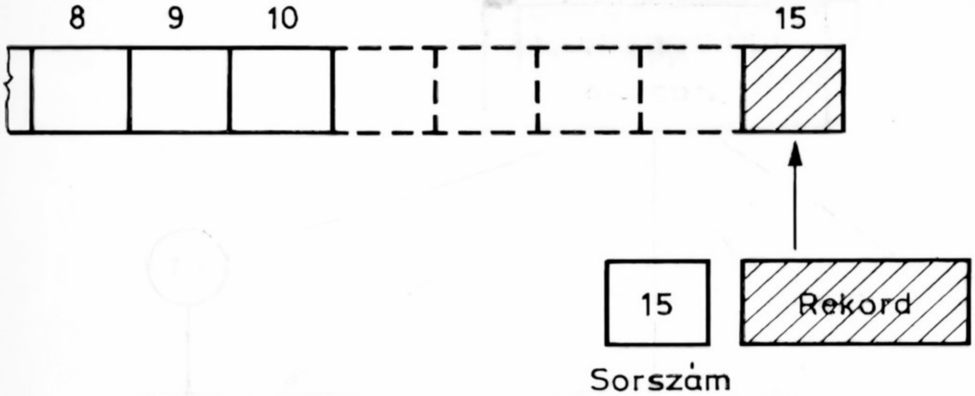
Ezekből az üres rekordokból áll tehát a relatív állományunk, amelyet a lemezkezelő az általunk megadott néven jegyez be a lemez tartalomjegyzékébe.

Mit jelent azonban az, hogy egy rekord üres? Nyilvánvalóan nem azt, hogy légüres térrel van tele. Az általunk üresnek nevezett rekordokat a lemezkezelő egységesen alapértelmezés szerinti karakterekkel tölti fel (22. ábra).

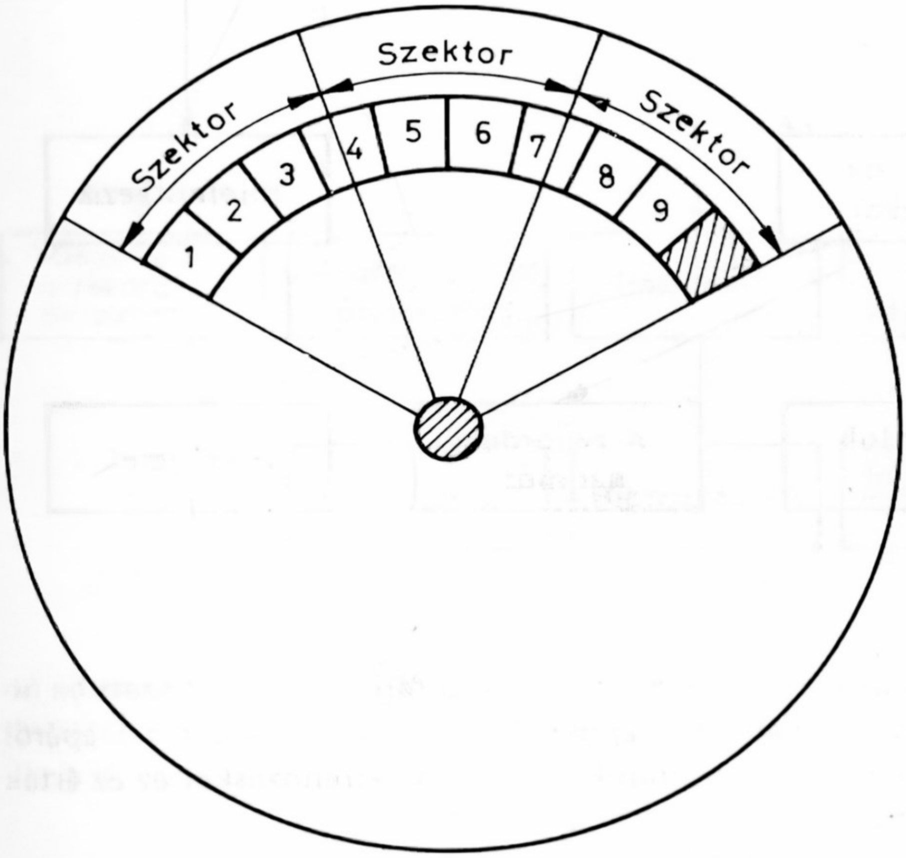
Ez úgy értendő, hogy a rekord első bájtjára hexadecimális FF, vagyis az egy bájton ábrázolható legnagyobb érték (high value) kerül. Ez a CHR$(255) karakternek, a π jelének felel meg. A rekord összes többi bájtja pedig bináris nullákat, vagyis az egy bájton ábrázolható legkisebb értéket (low value) tartalmazza. Ennek megfelelő karakter nincs a jelkészletben.

A relatív állomány létrehozásához szükséges információk tehát:

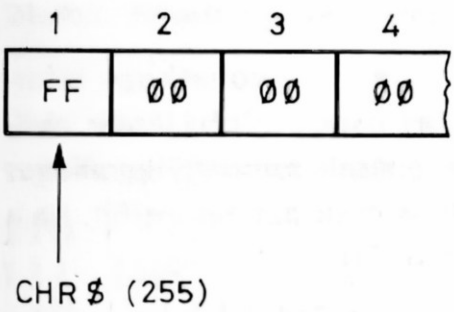
- az állomány neve,
- a rekordok mérete,
- a rekordok száma,



20. ábra. Automatikus bővítés

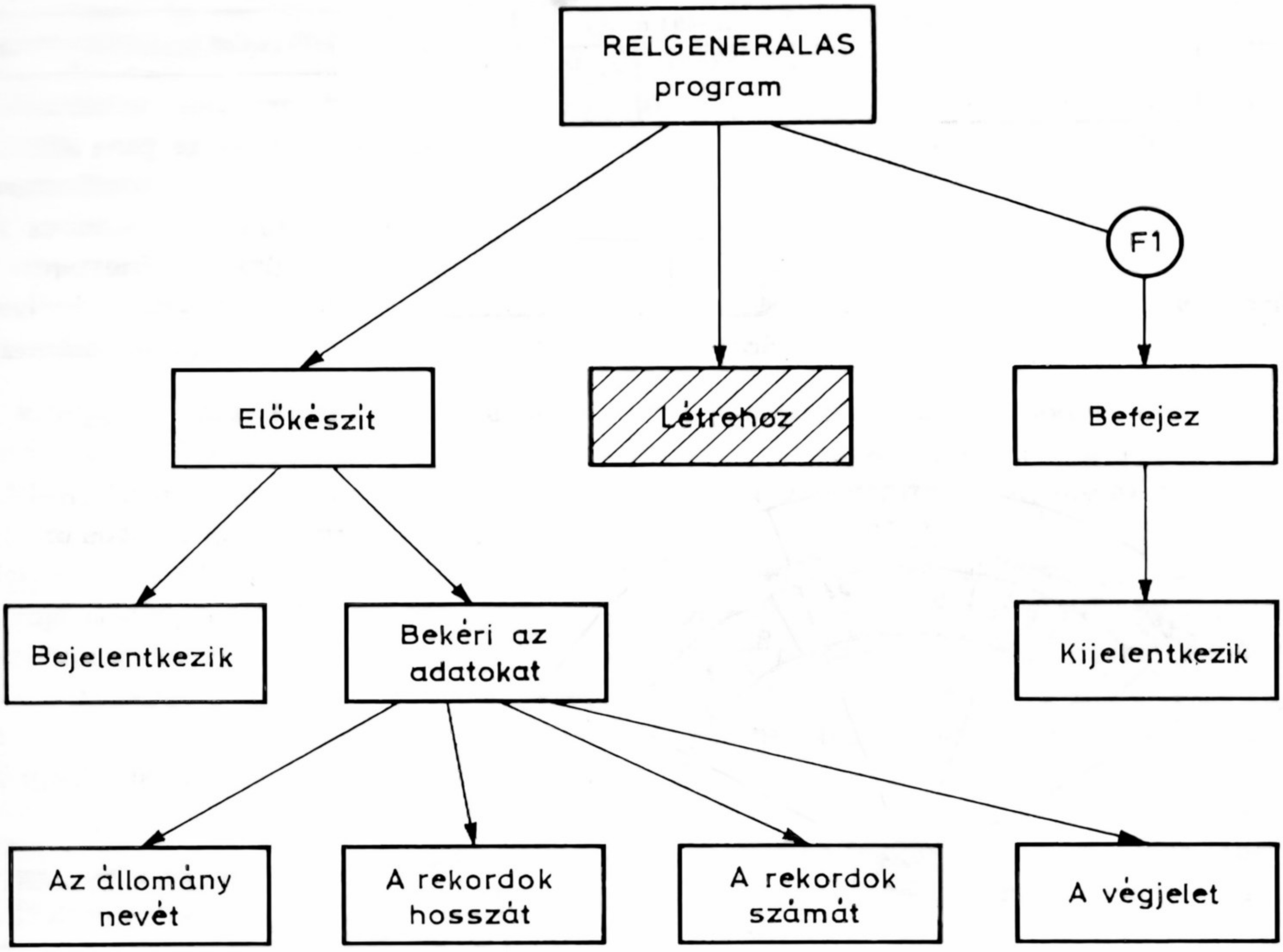


21. ábra. Az előkészített lemezterület



22. ábra. Üres rekord

amelyeket egy külön erre a célra írt létrehozandó programban célszerű megadnunk. Minthogy minden relatív állományt ugyanúgy kell létrehozni, érdemes olyan programot írnunk, amellyel tetszőleges relatív állományt generálhatunk (23. ábra).



23. ábra. Relatív állomány létrehozása I.

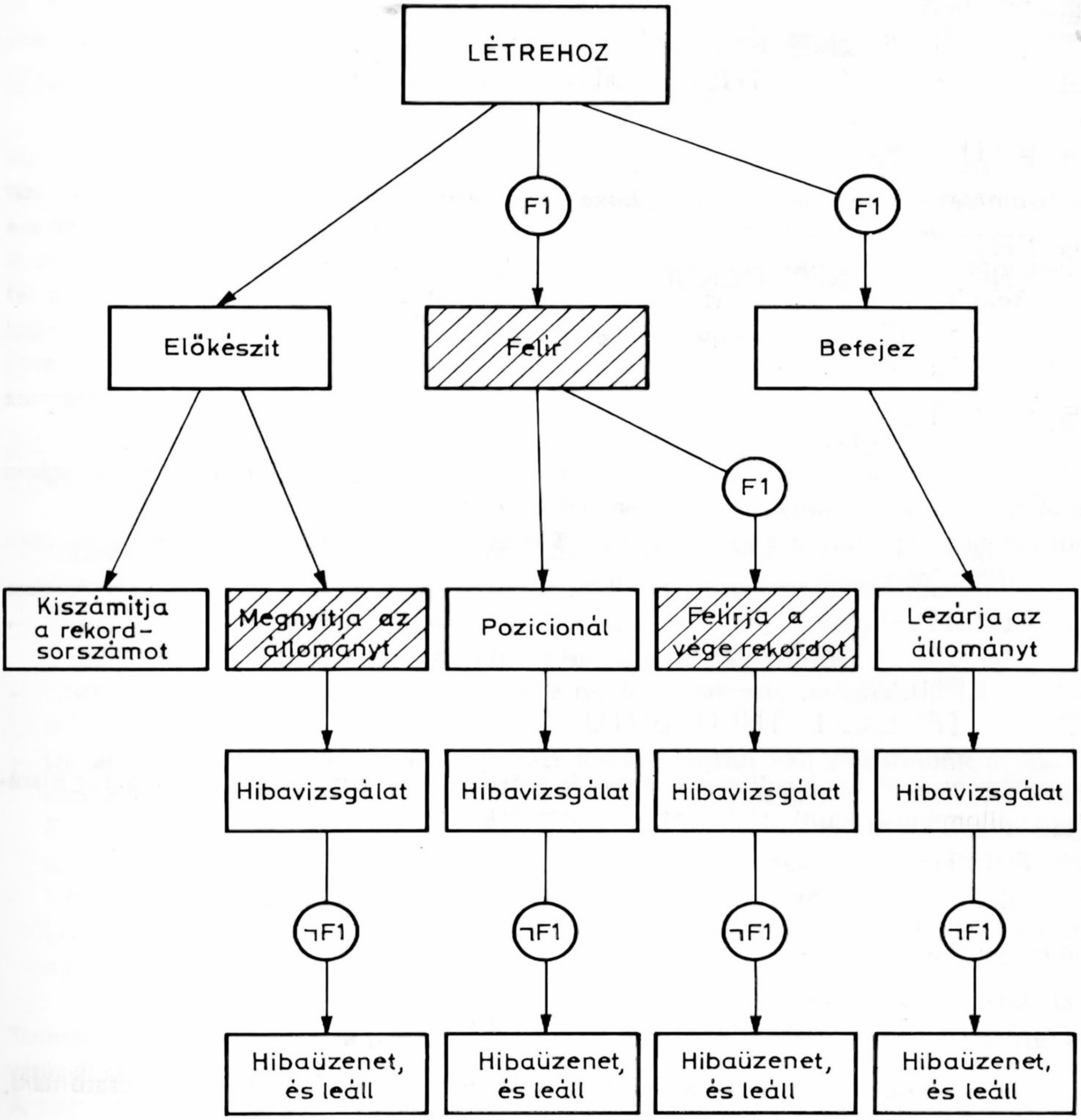
Ahol a befejezés csak akkor hajtható végre, ha a létrehozás során nem következett be hiba, azaz a H hibakód vagy kisebb 20-nál, vagy éppen 50. Az 50-es hibakód szerepéről később még lesz szó. Most elegendő annyit tudnunk róla, hogy létrehozáskor ez az érték valójában nem jelent hibát (24. ábra).

A létrehozás lényege egy rekord felvitele az állományba, természetesen megfelelő pozicionálás után. Ez határozza meg az állomány méretét.

A létrehozásban kulcsszerepe van a megnyitásnak, ahol az állományra vonatkozó információkat közöljük a lemezkezelővel. A megnyitástól kezdve az összes többi lépést nyilvánvalóan csak akkor érdemes végrehajtanunk, ha a korábbi lépések mindegyike sikeres volt. Ezért van szükség a gyakori hibavizsgálatra. Hibának itt is csak azt tekintjük, ha a parancscsatornáról beolvasott H hibakód 19-nél nagyobb, de nem 50.

A program először bejelentkezik:

```
101 PRINT "□":PRINT:PRINT
102 PRINT " ■                              ■"
103 PRINT " ■ RELATIV ALLOMANY LETREHOZASA ■"
104 PRINT " ■                              ■"
105 PRINT:PRINT
```



24. ábra. Relatív állomány létrehozása II. – LÉTREHOZ

Majd bekéri a relatív állomány F$ nevét, természetesen kizárva, hogy az üres legyen, vagy hogy 16 karakternél többet tartalmazzon:

```
110 PRINT
111 INPUT " ■*■ ALLOMANY  NEVE =";F$
112 L=LEN(F$)
113 :   IF L<1  THEN GOTO 111
114 :   IF L>16 THEN GOTO 111
```

Bekéri az R rekordhosszt, amely elvileg 1 és 254 közötti érték lehet (természetesen minden egyes rekord azonos hosszúságú a relatív állományban):

```
120 PRINT
121 INPUT " ■*■ REKORD  HOSSZA =";R
122 :  IF R<1   THEN GOTO 121
123 :  IF R>254 THEN GOTO 121
124 R=INT(R)
```

A rekordméret ismeretében a program bekéri a rekordok S számát:

```
130 PRINT
131 INPUT " ■*■ REKORDOK SZAMA  =";S
132 :  IF S<1        THEN GOTO 131
133 :  IF S>167132/R THEN GOTO 131
134 :  IF S>65535    THEN GOTO 131
135 S=INT(S)
```

A létrehozást csak akkor hajtja végre, ha a rekordméret függvényében a rekordok száma nem sérti a relatív állományokra kirótt korlátozásokat.

Végül meghatározhatjuk azt az egybájtos V$ jelet, amely az általunk felírt, egyetlen nem üres rekordba fog kerülni:

```
150 PRINT
151 INPUT " ■*■ VEGJEL          =";V$
152 L=LEN(V$)
153 :  IF L<>1 THEN GOTO 151
```

A program csak ekkor kezdi el a létrehozás előkészítését, amelynek során először kiszámítja az állomány általunk utolsónak tekintett rekordjának pozícióját:

```
200 RH=INT(S/256)
201 RL=S-RH*256
```

Majd megnyitja az állományt:

```
210 OPEN 15,8,15
220 OPEN 2,8,2,F$+",L,"+CHR$(R)
```

Minthogy a pozicionáláshoz és a hibalekérdezéshez szükség lesz a parancscsatornára, először is azt kell megnyitni.

Ezután következik magának a relatív állománynak a megnyitása létrehozásra, ami a szokásos módon kezdődik:

- Először megadjuk az állományazonosítót (2), a lemezegység számát (8) és az állomány számára lefoglalt adatcsatornát (2).
- Ezután közöljük az állomány nevét (F$).
- Majd egy L betű következik, amellyel meghatározzuk hogy most nem soros, hanem relatív állomány megnyitásáról van szó, éspedig létrehozásra.
- Végül bináris formában meg kell adnunk a rekordméretet (R).

Ezen információk hatására hozza létre a lemezkezelő a lemez tartalomjegyzékében a szükséges bejegyzéseket, valamint a relatív állomány későbbi kezeléséhez nélkülözhetetlen nyilvántartást.

A megnyitás sikerességét külön hibarutin vizsgálja meg, amely a megadott paraméterek alapján a hiba L helyét és a sikertelen L$ funkciót is megadja:

```
230 L=220: L$="OPEN ": GOSUB 1010
```

Az OPEN utasításban is látható, hogy a numerikus értéket a CHR$ függvénnyel alakíthatjuk át bináris formára. Ehhez a konverzióhoz, ellentétben a rekordsorszámmal, a rekordméretet nem kell felbontanunk, hiszen az értéke 256-nál mindig kisebb.
A sikeres megnyitás azonban nem hozza létre az előkészített lemezterületet, azaz nem írja fel a megfelelő számú üres rekordot a lemezre. Most következik tehát az állomány tényleges létrehozása. Minthogy az állomány méretét a benne tárolandó rekordok száma határozza meg, a program pozicionál az általunk korábban megadott maximális R rekordsorszámra:

```
240 PRINT#15,"P"CHR$(2)CHR$(RL)CHR$(RH)CHR$(1)
250 L=240: L$="POSIT": GOSUB 1010
```

A pozicionáló információkat a random állományoknál megismert állománykezelő információkhoz hasonlóan a parancscsatornára szóló PRINT utasításban kell kiadnunk:
– Először a pozicionálást meghatározó kulcsszó jön, amelynek eredeti alakja (POSITION) helyett gyakran a rövidített formáját (P) használjuk.
– Ezután következik a relatív állományhoz rendelt adatcsatorna száma (2), bináris alakban megadva.
– Ugyancsak bináris alakban, két egymást követő bájton kell megadnunk a rekord R sorszámát, mégpedig először mindig az alsó (RL), utána pedig a felső (RH) bájtot. Ezeknek egyike sem hagyható el, még akkor sem, ha az értéke nulla lenne. A két bájt együttesen határozza meg, hogy melyik rekordot kell a lemezkezelőnek létrehoznia.
– Végül szintén bináris formában meg kell határoznunk a rekord azon bájtpozícióját (1), amelyen a rekord feltöltésének kezdődnie kell. Ez az információ sem hiányozhat a pozicionálásból.

Természetesen a program a pozicionálás sikerességéről is meggyőződik a hibarutin behívásával, és csak akkor megy tovább, ha nem volt hiba.
A pozicionálás hatása az lesz, hogy a lemezkezelő megkeresi a lemezen a megadott rekordot. Minthogy az állomány még nem létezik, nem fogja megtalálni. (A hibarutin ilyenkor **50**-es hibakódot és **"RECORD NOT PRESENT"**, azaz „a rekord az állományban nem található" hibaüzenetet ad. Esetünkben ez azonban nem hiba.) A lemezkezelő felkészül arra, hogy a rekordot létre kell hoznia, de még nem teszi meg, mivel nem tudja, hogy létre akarjuk-e hozni a rekordot, vagy csak lekérdezni szerettük volna. Azt sem tudja még, hogy ha létre kell hoznia, akkor milyen adatokat tegyen bele.
A programban a pozicionálást egy lemezreíró (PRINT) utasítás követi. Ebből a lemezkezelő számára nyilvánvalóvá válik, hogy nem olvasni, hanem írni akarunk; a rekordot tehát létre kell hoznia, és az utasításban felsorolt változók a rekord tartalmát is meghatározzák.

```
270 PRINT#2,V$
280 L=270: L$="PRINT": GOSUB 1010
```

A rekord tartalma a létrehozás szempontjából közömbös, akármilyen információt tárolhatunk benne. Mi általános generáló programot kívántunk létrehozni, ezért valamilyen semleges, illetve az állomány végére utaló V$ adatot adtunk meg. Ez a megadott sorszámú rekordba a pozicionáláskor beállított bájtra kerül; vagyis esetünkben az első bájtra.

A PRINT utasítás hatása nem egyszerűen a rekord felírása lesz. A lemezkezelő ugyanis az 50-es hibakód alapján **(RECORD NOT PRESENT)** már a pozicionálás óta tudja, hogy a rekord számára nincs hely az állományban. Úgy viselkedik tehát, mintha az állományt kiterjesztené, azzal az eltéréssel, hogy most az OPEN utasításból tudja, hogy létrehozásról szó, így nem az utolsó létező rekordtól, hanem az első rekordtól tölti fel az állományt üres rekordokkal, egészen a megadott sorszámig; majd felírja a PRINT utasításban megadott adat tartalmával feltöltött utolsó rekordot is. Ezzel létrejön a relatív állomány kezeléséhez szükséges lemezterület.

A művelet sikerességéről illik a hibakezelő rutin aktivizálásával meggyőződni. Ha a program a létrehozás eddigi menetét kielégítőnek találja, lezárja az állományt, a parancscsatornával együtt:

```
310 CLOSE 2
320 CLOSE 15
```

A lezárás hatására vezeti rá a lemezkezelő az állományhoz felhasznált blokkok foglalt jelzését a lemezen tárolt lemeztérképre, és ha szükséges, karbantartja a relatív állomány blokkokról készített saját nyilvántartását is.

A program ezek után kijelentkezik, és leáll:

```
910 PRINT:PRINT:PRINT
920 PRINT " ▣               ▦"
930 PRINT " ▣ LETREHOZVA ▦"
940 PRINT " ▣               ▦"
999 END
```

A gyakran hívott hibarutin lekérdezi a parancscsatornát, és hiba esetén a kapott paraméterekből megfelelő hibaüzenetet állít össze, megadva a lemezkezelő H hibakódját, H$ hibaüzenetét, a hiba előfordulásának L sorát, és a hibásan végrehajtódott vagy megszakadt művelet kulcsszavának L$ rövidítését; majd lezárja a csatornákat, és leállítja a programot:

```
1010 INPUT#15,H,H$,T,B
1020 IF H<20 THEN GOTO 1999
1030 IF H=50 THEN GOTO 1999
1040 :   K$=STR$(H)+" : "+H$+" "
1050 :   S$=""
1051 :   FOR I=1 TO LEN(K$)
1052 :   :   S$=S$+" "
1059 :   NEXT I
1060 :   PRINT:PRINT:PRINT
1070 :   PRINT " ▣";S$;"▦"
1080 :   PRINT " ▣";K$;"▦"
1090 :   PRINT " ▣";S$;"▦"
1100 :   CLOSE 2
1110 :   CLOSE 15
```

```
1120 :  PRINT
1130 :  PRINT " SOR=";L,L$
1140 :  PRINT
1199 :  STOP
1999 RETURN
```

A programot begépelése után először mentsük ki RELGENERALAS néven. Utána akár ki is próbálhatjuk. Hozzunk létre egy "PROBA" nevű relatív állományt 10 darab, egyenként 10 karakteres rekorddal; a végjel pedig legyen csillag.

```
RELATIV ALLOMANY LETREHOZASA

 ALLOMANY  NEVE =? PROBA
 REKORD  HOSSZA =? 10
 REKORDOK SZAMA =? 10
 VEGJEL         =? *

LETREHOZVA
```

Az állomány néhány másodperc alatt létrejön. Ha lekérjük és a lemez tartalomjegyzékét kilistázzuk a képernyőre, láthatjuk, hogy az állomány "PROBA" néven be van jegyezve, két blokkot foglal el, és az eddig megszokott PRG vagy SEQ minősítéssel ellentétben, a megkülönböztető jele REL.

A RELATÍV ÁLLOMÁNY BÁJTONKÉNTI OLVASÁSA

Jobban megértjük a relatív állományok kezelésének elvét, ha megnézzük a létrejött állomány tartalmát.

A relatív állomány rekordjai sorszám szerint követik egymást, ezért kézenfekvő az ilyen állományok soros feldolgozása: egyszerűen a sorszámok mentén végighaladva beolvashatjuk és feldolgozhatjuk az összes rekordot. Természetesen a soros feldolgozás itt is közvetlen elérések láncolatával valósul meg, mint ahogyan az a random állománynál is történt.

Vegyük a legegyszerűbb feldolgozást, amely végigolvassa az állományt, és minden rekordból kiír annyi bájtot, amennyit meghatározunk. Nehezítsük ezt azzal, hogy a kiírás ne csak a teljes állományra, hanem tetszőleges rekordjától ugyancsak tetszőleges rekordjáig tartó bármely intervallumára is végrehajtható legyen (25. ábra).

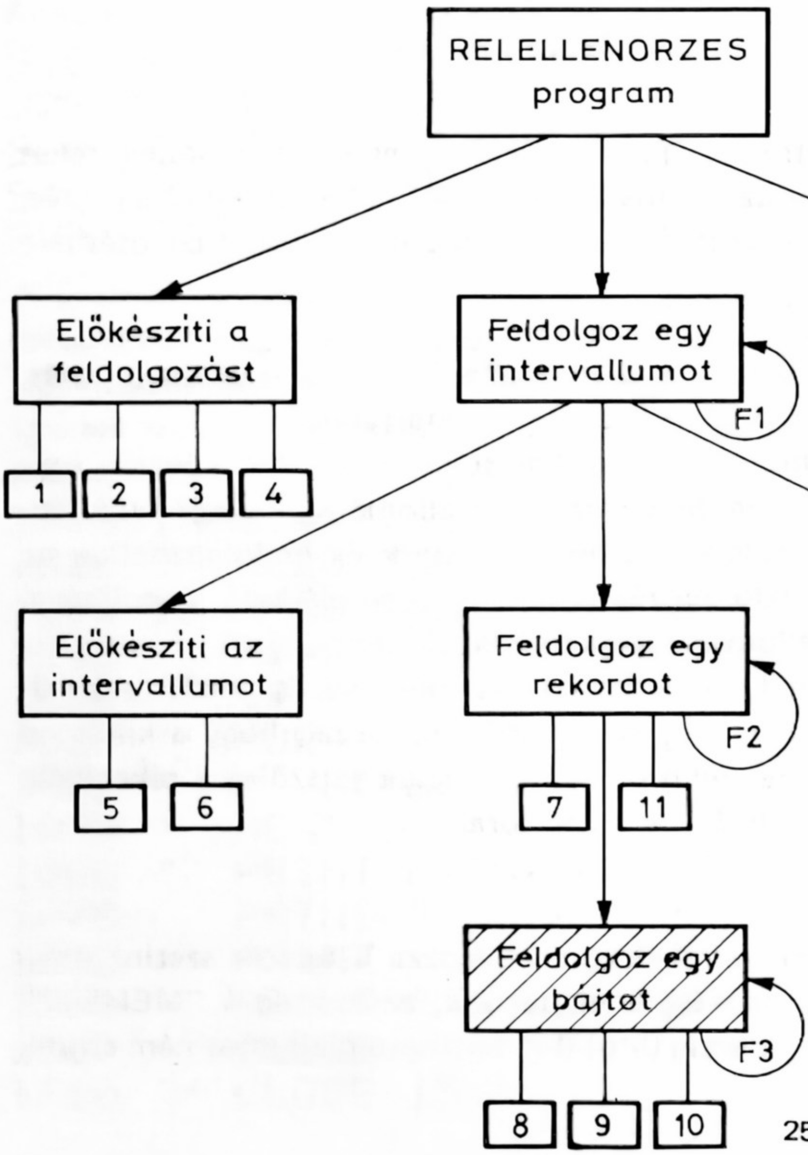
A feltételek:

– F1: A program tetszőleges sorrendben mindaddig feldolgozza a tetszés szerint megadott intervallumokat, amíg azt szükségesnek tartjuk, azaz amíg a "MEHET?" kérdésre a V$ válaszunk="I", vagy amíg 0-tól 0-ig tartó intervallumot nem adunk meg.

— F2: Az intervallumhoz tartozó rekordok mindegyikét feldolgozza, az A alsó sorszámtól az F felső sorszámig egyesével, illetve amíg olvasási hiba nem fordul elő.
— F3: A rekordon belüli bájtok feldolgozása 1-től egyesével a megadott N bájtig tart.

A program résztevékenységei pedig a következők:

— 1: bejelentkezik,
— 2: bekéri az állomány nevét,
— 3: bekéri a beolvasandó bájtok számát,
— 4: megnyitja az állományt,
— 5: bekéri az intervallumot,
— 6: fejlécet készít,
— 7: a rekordsorszámot bináris formára konvertálja,
— 8: a rekord soron következő bájtjára pozicionál,
— 9: hibavizsgálatot hajt végre,
— 10: beolvassa a bájtot,
— 11: kiírja a rekord tartalmát,
— 12: láblécet készít,
— 13: megkérdezi, hogy jöhet-e a következő intervallum,
— 14: kijelentkezik,
— 15: lezárja az állományt.



25. ábra. Relatív állomány bájtonkénti olvasása

A program egyszerű. A bejelentkezésben semmi szokatlan nincs:

```
101 PRINT "□":PRINT:PRINT
102 PRINT "                                              ■"
103 PRINT "  RELATIV ALLOMANY ELLENORZESE ■"
104 PRINT "                                              ■"
105 PRINT:PRINT
```

Az állomány F$ nevének bekérésekor a formailag hibás neveket nem fogadja el:

```
110 PRINT
120 INPUT "■ ALLOMANY=";F$
130 L=LEN(F$)
140 :    IF 1>L OR L>16 THEN GOTO 120
```

Bekéri a rekordokból beolvasandó bájtok N számát:

```
150 PRINT:PRINT
160 PRINT
161 PRINT "■ EGY REKORDBOL OLVASANDO";
162 INPUT "    1   ■ BAJT■■■■■■■■■■■■■■■■";N$
163 N=VAL(LEFT$(N$,3))
164 :    IF 1>N OR N>253 THEN GOTO 160
```

A relatív állomány megnyitása csak létrehozáskor bonyolult. Ugyanis az ott megadott információkat a lemezkezelő nyilvántartja, így ha egy már létező relatív állományt akarunk megnyitni, elegendő csak a nevét megadnunk:

```
210 OPEN 15,8,15
220 OPEN 2,8,2,F$
```

A parancscsatornára természetesen itt is szükség lesz, tehát azt is meg kell nyitni.

Megjegyezzük, hogy egy létező relatív állomány írásra és olvasásra egyaránt használható, így nem szükséges a soros állományoknál szokásos READ vagy WRITE információt megadni, sőt egyenesen **tilos!** A lemezkezelő éppen onnan tudja meg, hogy egy létező relatív állomány megnyitására kell felkészülnie, hogy az OPEN utasításban az állomány nevén kívül más információt nem talál. Ennek megfelelően jár is el.

FIGYELEM!

Noha a relatív és soros állományok, valamint a programok megkülönböztető jelzést kapnak, ami a tartalomjegyzék listáján REL, SEQ, PFRG formában jelenik meg a lemezkezelő csak a nevük szerint különbözteti meg az állományokat; így a lemezen egyidejűleg nem lehet "PROBA" nevű relatív állományunk és "PROBA" nevű programunk. Ez soros állományokra is vonatkozik.

A feldolgozási intervallum meghatározza, hogy mely A rekordtól mely F rekordig tartson a rekordok bájtjainak kiírása. Bekérésekor a program ellenőrzi, hogy az intervallum határaiként megadott rekordsorszámok formailag helyesek-e, és ha nem, új intervallumot kér:

```
310 PRINT "□": PRINT
320 PRINT "■ INTERVALLUM "
330 :    PRINT
```

```
340 :   INPUT "  REKORDTOL =  0■■■■";A
350 :   PRINT
360 :   INPUT "  REKORDIG  =  0■■■■";F
370 : :   IF A=0 AND F=0      THEN GOTO 810
380 : :   IF A<1 OR  A>65535 THEN GOTO 330
390 : :   IF F<A OR  F>65535 THEN GOTO 330
391 : :   IF F-A > 20         THEN GOTO 330
```

A program 20 rekordnál hosszabb intervallumot nem fogad el, hiszen kiírása már nem férne el a képernyőn. Minthogy egy intervallum kiírása után kérhetjük a következőt, ez a 20-as határ valójában semmiféle korlátozást nem jelent.

Megjegyezzük, hogy a program 0-tól 0-ig tartó intervallummal is leállítható, ha rájövünk, hogy tévesen adtuk meg az állomány F$ nevét, vagy hogy a "MEHET?" kérdésre meggondolatlanul adtunk igenlő V$ választ.

Az egymás után lekérdezett intervallumok tetszőleges sorrendben követhetik egymást, sőt lehetnek köztük átfedők is. Annak sincs akadálya, hogy valamely intervallum csupán egyetlen rekordból álljon.

A fejléc a lehető legegyszerűbb, hogy ne foglaljon el sok helyet a képernyőn:

```
398 :   PRINT "□"
399 :   PRINT " ──────────┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬┬"
```

Az intervallumot feldolgozó ciklus elindítása után a program az RS rekordszámot a pozicionáláshoz szükséges bináris formára konvertálja, és törli a rekordtartalom tárolására használt B$ változót:

```
410 :   FOR RS=A TO F
420 : :   RH=INT(RS/256)
430 : :   RL=RS-RH*256
440 : :   B$=""
```

Minden rekordból annyi N bájtot olvas be a program, ahányat megadtunk neki. Ehhez egy ciklust indít, amelyben először a rekord soron következő K bájtjára pozicionál, majd hibavizsgálatot hajt végre:

```
500 : :   FOR K=1 TO N
510 : : :   PRINT#15,"P"CHR$(2)CHR$(RL)CHR$(RH)CHR$(
520 : : :   GOSUB 1010 :REM==> HIBAVIZSGALAT
530 : : :   IF H>=20 THEN GOTO 710
```

Sikertelen pozicionálás esetén kilép a ciklusból. Ha viszont nem volt hiba, beolvassa a bájtot, és bejegyzi a rekordtartalmat tároló változóba:

```
540 : : : :   GET#2,C$
550 : : : :   IF C$=CHR$(13) THEN C$="♦"
560 : : : :   IF C$="" THEN C$=":"
570 : : : :   B$=B$+C$
599 : :   NEXT K
```

Az olvasást a már ismert GET utasítással hajtja végre. A képernyőn önkényesen kiválasztott C$ karakterek jelennek meg azon bájtok helyett, amelyek meg nem jeleníthető karakterekkel kifejezhető információt tartalmaznak; vagyis a program a RETURN jelet sarkára állított négyzetre, a bináris nullákat pedig kettőspontra cseréli ki (természetesen csak a képernyőn, a rekordban nem!).
A bájtok beolvasása után a program kiírja az R$ rekordsorszámot, valamint a rekord általunk megadott első N bájtjának B$ tartalmát:

```
610 :   :  R$=RIGHT$("      "+STR$(RS),6)
620 :   :  PRINT R$;" = ";B$
699 :  NEXT RS
```

Az intervallum feldolgozásának befejeztével a program láblécet készít:

```
700 :  PRINT " ──────────┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴"
```

Majd megkérdezi, hogy jöhet-e a következő intervallum. Ugyanezt teszi, ha az intervallum feldolgozása közben hiba fordult elő:

```
710 :  PRINT
720 :  INPUT "▒▒■ MEHET =I/N=  I█▒▒█";V$
730 :  IF V$<>"I" THEN GOTO 810
799 GOTO 310
```

Igenlő V$ válasz esetén visszatér az intervallum bekérésére. Egyébként pedig kijelentkezik, lezárja az állományt, és leáll:

```
810 PRINT:PRINT"▒▒■ VEGE"
900 REM ---------------------------------------- LEZARAS
910 CLOSE 2
920 CLOSE 15
999 END
```

A hibavizsgálatot külön szubrutin hajtja végre, amely lekérdezi a parancscsatornát, és hiba esetén megadja az R$ rekordsorszámot, a K bájtpozíciót, valamint a lemezkezelő H$ hibaüzenetét:

```
1010 INPUT#15,H,H$,S,B
1020 IF H<20 THEN GOTO 1999
1030 :  PRINT " ──────────┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴┴"
1040 :  PRINT " ▒";RS;K;" - ";H$;"! ■"
1999 RETURN
```

Ezzel kapcsolatban megjegyezzük, hogy a program (azaz a programozó) optimista, mivel a hibavizsgálatot csak a pozicionálás után hajtja végre, az olvasás után azonban nem. Ez az optimizmus azonban megalapozott, mivel a sikeres pozicionálás után már igen kicsi a valószínűsége a bájtonkénti olvasás sikertelenségének.
A begépelt programot mentsük ki RELELLENORZES néven a lemezünkre, mert később még szükségünk lesz rá.
Ezután kipróbálhatjuk. A program kérdésére állománynévként a "PROBA" nevet adjuk meg, a beolvasandó bájtok száma pedig 10 legyen:

RELATIV ALLOMANY ELLENORZESE

```
:: ALLOMANY=? PROBA

:: EGY REKORDBOL OLVASANDO? 10 BAJT
```

Minthogy a lemezen PROBA néven tárolt relatív állományunk tízbájtos rekordokból áll, a kiíráskor a teljes rekordtartalmat meg fogjuk kapni.
Az első lekérdezett intervallum 1-től 20-ig tartson.

```
:: INTERVALLUM
   REKORDTOL =? 1
   REKORDIG  =? 20
```

Rövidesen megjelenik a képernyőn az első 20 rekord tartalma:

```
      1 = π:::::::::
      2 = π:::::::::
      3 = π:::::::::
      4 = π:::::::::
      5 = π:::::::::
      6 = π:::::::::
      7 = π:::::::::
      8 = π:::::::::
      9 = π:::::::::
     10 = *◆::::::::
     11 = π:::::::::
     12 = π:::::::::
     13 = π:::::::::
     14 = π:::::::::
     15 = π:::::::::
     16 = π:::::::::
     17 = π:::::::::
     18 = π:::::::::
     19 = π:::::::::
     20 = π:::::::::
```

```
:: MEHET =I/N=? I
```

Először is látható, hogy az első kilenc rekord üres: az első bájton a legmagasabb értéket, a CHR$(255) karaktert, azaz a π jelet tartalmazza, az összes többi bájton pedig bináris nulla van.

A tizedik rekord azt tartalmazza, amit a létrehozáskor beleírtunk: az első bájton csillagot, majd a felíráskor keletkezett RETURN jelet, azaz a CHR$(13) karaktert. A többi bájton bináris nullák vannak.

Itt azonban nem ér véget az állomány, noha csak 10 rekord létrehozására adtunk parancsot. A 11. rekordtól újra üres rekordok következnek. Ennek az a már ismert tény az oka, hogy az állomány létrehozásakor a lemezkezelő a megkezdett szektort mindenképpen végig feltölti. Egy blokkban, azaz egy szektoron a tízbájtos rekordokból 25 fér el, ezért várhatóan ennyi rekord lesz az imént létrehozott állományban. Meggyőződhetünk róla, ha a "MEHET?" kérdésre "I"-vel válaszolunk, majd a következő intervallum határaiként 21 és 40 értéket adunk meg.

```
21 = π: : : : : : : : :
22 = π: : : : : : : : :
23 = π: : : : : : : : :
24 = π: : : : : : : : :
25 = π: : : : : : : : :
```

[illegible]

```
3 MEHET =I/N=? N
```

Valóban, a 25. rekordig megjelennek az üres rekordtartalmak, de a 26. rekord első bájtjánál **RECORD NOT PRESENT**, azaz a rekord az állományban nem található hibaüzenetet kapunk. Ehhez az üzenethez az 50-es hibakód tartozik. Tanulságként megjegyezhetjük, hogy soros végigolvasáskor az állomány tényleges végét erről a hibakódértékről ismerhetjük fel. A létrehozáskor általunk felvitt rekordnak e tekintetben nincs szerepe.

A példa azonban arra is felhívja a figyelmünket, hogy ha egy rekordba adatot viszünk, célszerű az első bájtra CHR$(255)-től eltérő értéket írni, mert így a feltöltött rekordok könnyen megkülönböztethetők lesznek az üresektől.

A programot a hibaüzenet után nem kell leállítanunk, kérhetünk újabb intervallumot. Esetünkben ennek nincs értelme, hiszen az egész állományt láttuk. Tehát állítsuk le a programot a "MEHET?" kérdésre adott tagadó válasszal.

Ha kívánjuk, kipróbálhatjuk a programot úgy is, hogy a beolvasandó bájtok számaként nem 10, hanem ennél kevesebb értéket adunk meg.

FIGYELEM!

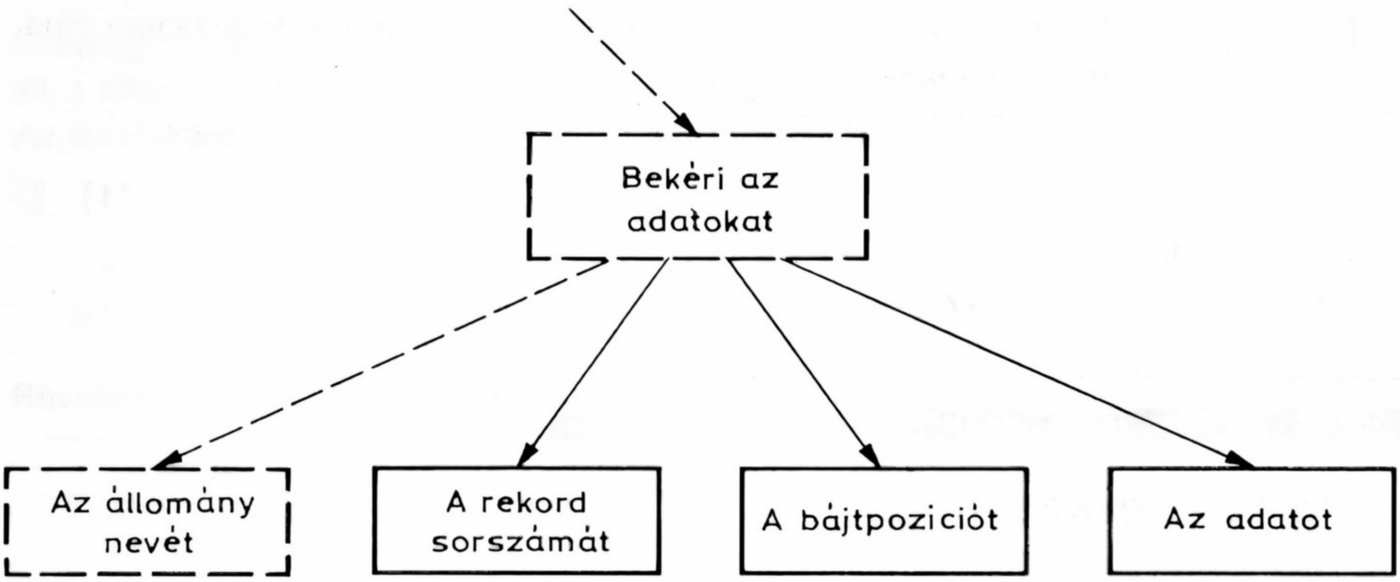
Az aktuális rekordhossznál (azaz esetünkben 10-nél) nagyobb érték esetén már az első rekordnál **OVERFLOW IN RECORD** azaz „rekordhatár túllépése" hibaüzenetet fogunk kapni. Próbáljuk ki, mondjuk 12-vel!

Megjegyezzük, hogy ha a 3-as tevékenységet, azaz a beolvasandó bájtok számának bekérését egy szinttel alacsonyabban, az intervallum feldolgozásának előkészítésekor hajtjuk végre, mondjuk az 5-ös és a 6-os tevékenységek között, akkor az egyes intervallumokat különböző bájthosszúságban kérdezhetjük le, a program újraindítása nélkül. Ehhez csak a 150–164 sorokat kell áthelyeznünk a 392–397 sorokra, természetesen a GOTO 160 utasítást GOTO 393-ra cserélve.

A RELATÍV ÁLLOMÁNY FELTÖLTÉSE

A relatív állomány rekordjainak adatokkal való feltöltése eltér az eddig megismert rekordkezelési módoktól. Ezért érdemes egy mintaprogramot írnunk e művelet sajátosságainak kipróbálására.

A program a RELGENERALAS létrehozó programból könnyen előállítható. Módosítanunk kell a rekordok hosszát és számát, a végjel helyett pedig a rekordsorszámot, a bájtpozíciót és a rekordba betöltendő adatot kell bekérnünk (26. ábra).



26. ábra. Relatív állomány írása

Természetesen a rekordsorszám és a bájtpozíció annak a rekordnak a sorszámát, illetve a rekordon belül annak a bájtnak a pozícióját jelenti, amelyre a kérdéses adatot fel akarjuk vinni.

Tehát töltsük be a RELGENERALAS programot, majd hajtsuk végre a szükséges módosításokat:

Töröljük az alábbi sorokat:

```
120 REM ------TOROLVE------
121 REM ------TOROLVE------
122 REM ------TOROLVE------
123 REM ------TOROLVE------
124 REM ------TOROLVE------
133 REM ------TOROLVE------
150 REM ------TOROLVE------
151 REM ------TOROLVE------
152 REM ------TOROLVE------
153 REM ------TOROLVE------
```

Szúrjuk be a következőket:

```
140 PRINT
141 INPUT " **■ BAJTPOZICIO      =";P
142 IF P<0     THEN GOTO 141
143 IF P>253   THEN GOTO 141
```

```
144 P=INT(P)
160 PRINT
161 INPUT " ▓*■ FELIRANDO ADAT =";A$
```

Ezeket a sorokat pedig módosítsuk:

```
102 PRINT " ▓                            ■"
103 PRINT " ▓ RELATIV ALLOMANY IRASA ■"
104 PRINT " ▓                            ■"
131 INPUT " ▓*■ REKORDSORSZAM  =";S
220 OPEN 2,8,2,F$
240 PRINT#15,"P"CHR$(2)CHR$(RL)CHR$(RH)CHR$(P)
270 PRINT#2,A$
920 PRINT " ▓           ■"
930 PRINT " ▓  FELIRVA  ■"
940 PRINT " ▓           ■"
```

A teljes program ezek után ilyen lesz:

```
1 REM:    RELATIVIRAS
2 REM:    ------------
3 REM:
100 REM ----------------------------------------ADATBEKERES
101 PRINT "▯":PRINT:PRINT
102 PRINT " ▓                            ■"
103 PRINT " ▓ RELATIV ALLOMANY IRASA ■"
104 PRINT " ▓                            ■"
105 PRINT:PRINT
110 PRINT
111 INPUT " ▓*■ ALLOMANY  NEVE =";F$
112 L=LEN(F$)
113 :   IF L<1  THEN GOTO 111
114 :   IF L>16 THEN GOTO 111
130 PRINT
131 INPUT " ▓*■ REKORDSORSZAM  =";S
132 :   IF S<0      THEN GOTO 131
134 :   IF S>65535 THEN GOTO 131
135 S=INT(S)
140 PRINT
141 INPUT " ▓*■ BAJTPOZICIO     =";P
142 :   IF P<0      THEN GOTO 141
143 :   IF P>253    THEN GOTO 141
144 P=INT(P)
160 PRINT
161 INPUT " ▓*■ FELIRANDO ADAT =";A$
199 REM ----------------------SORSZAMKONVERTALAS
200 RH=INT(S/256)
201 RL=S-RH*256
```

```
209 REM ------------------------------------ MEGNYITAS
210 OPEN 15,8,15
220 OPEN 2,8,2,F$
230 L=220: L$="OPEN ": GOSUB 1010
239 REM ---------------------------------POZICIONALAS
240 PRINT#15,"P"CHR$(2)CHR$(RL)CHR$(RH)CHR$(P)
250 L=240: L$="POSIT": GOSUB 1010
269 REM ----------------------------------------IRAS
270 PRINT#2,A$
280 L=270: L$="PRINT": GOSUB 1010
300 REM -------------------------------------LEZARAS
310 CLOSE 2
320 CLOSE 15
900 REM ------------------------------- KIJELENTKEZES
910 PRINT:PRINT:PRINT
920 PRINT " ▣                  █"
930 PRINT " ▣  FELIRVA  █"
940 PRINT " ▣                  █"
999 END
1000 REM ============================HIBAVIZSGALAT
1010 INPUT#15,H,H$,T,B
1020 IF H<20 THEN GOTO 1999
1030 IF H=50 THEN GOTO 1999
1040 :   K$=STR$(H)+" : "+H$+" "
1050 :   S$=""
1051 :   FOR I=1 TO LEN(K$)
1052 :   :  S$=S$+" "
1059 :   NEXT I
1060 :   PRINT:PRINT:PRINT
1070 :   PRINT " ▣";S$;"█"
1080 :   PRINT " ▣";K$;"█"
1090 :   PRINT " ▣";S$;"█"
1100 :   CLOSE 2
1110 :   CLOSE 15
1120 :   PRINT
1130 :   PRINT " SOR=";L,L$
1140 :   PRINT
1199 :   STOP
1999 RETURN
```

Ezzel kész is a mintaprogram, amely lehet, hogy nem a legkényelmesebben, de igen egyszerűen használható a relatív állomány rekordjaiba történő adatbevitel kipróbálására.

Felhívjuk a figyelmet az OPEN utasításra. Látható, hogy az olvasáshoz hasonlóan íráskor sincs szükségünk másra, mint az állomány nevének megadására – ha az állomány már létezik.

Mentsük ki a programot RELIRAS néven. Majd próbáljuk ki a lemezen már létező "PROBA" állományunkon az alábbi adatokkal (a programot természetesen minden egyes adat felírása után újraindítva):

```
ALLOMANY  NEVE =? PROBA
REKORDSORSZAM  =? 2
BAJTPOZICIO    =? 1
FELIRANDO ADAT =? AAAA

ALLOMANY  NEVE =? PROBA
REKORDSORSZAM  =? 4
BAJTPOZICIO    =? 4
FELIRANDO ADAT =? BBBB

ALLOMANY  NEVE =? PROBA
REKORDSORSZAM  =? 6
BAJTPOZICIO    =? 8
FELIRANDO ADAT =? CCCC

51 : OVERFLOW IN RECORD
SOR= 270            PRINT

BREAK IN 1199
```

Itt **OVERFLOW IN RECORD,** azaz rekordhatár túllépése hibaüzenetet kapunk, hiszen a negyedik C karakternek már a 11. bájtpozícióra kellene kerülnie, a rekord viszont csak tízbájtos. Ne ijedjünk meg, indítsuk újra a programot, és folytassuk az adatfelvitelt:

```
ALLOMANY  NEVE =? PROBA
REKORDSORSZAM  =? 8
BAJTPOZICIO    =? 1
FELIRANDO ADAT =? DDDD

ALLOMANY  NEVE =? PROBA
REKORDSORSZAM  =? 8
BAJTPOZICIO    =? 7
FELIRANDO ADAT =? EEEE
```

```
51 : OVERFLOW IN RECORD
 SOR= 270          PRINT

BREAK IN 1199
```

Itt is **OVERFLOW IN RECORD** hibaüzenetet kapunk, mert a rekordba nemcsak az adatnak, hanem a RETURN jelnek is el kell férnie. A hibát tehát **nem** az okozza, hogy ugyanabba a rekordba akartunk adatot írni, amelyikbe az előbb már írtunk. Ez ugyanis megengedett, és a rekordok közvetlen elérése miatt lehetséges is.

De menjünk tovább:

```
ALLOMANY  NEVE =? PROBA
REKORDSORSZAM  =? 10
BAJTPOZICIO    =? 3
FELIRANDO ADAT =? FFFF

ALLOMANY  NEVE =? PROBA
REKORDSORSZAM  =? 12
BAJTPOZICIO    =? 6
FELIRANDO ADAT =? GGGG

ALLOMANY  NEVE =? PROBA
REKORDSORSZAM  =? 14
BAJTPOZICIO    =? 6
FELIRANDO ADAT =? HHHH

ALLOMANY  NEVE =? PROBA
REKORDSORSZAM  =? 14
BAJTPOZICIO    =? 1
FELIRANDO ADAT =? IIII

ALLOMANY  NEVE =? PROBA
REKORDSORSZAM  =? 16
BAJTPOZICIO    =? 1
FELIRANDO ADAT =? JJJJ
```

```
:: ALLOMANY  NEVE =? PROBA
:: REKORDSORSZAM  =? 16
:: BAJTPOZICIO    =? 4
:: FELIRANDO ADAT =? KKKK
```

Most már elegendő adatunk van az állományban, amelynek nézzük meg a tartalmát.
Töltsük be a RELELLENORZES programot a tárba, és indítsuk el a "PROBA" állományra, rekordonként 10 bájt olvasását előírva az 1.-től a 17. rekordig terjedő intervallumon belül.
Ekkor a képernyőn megjelenik az állomány tartalma:

```
      1 = π:::::::::
      2 = AAAA♦:::::
      3 = π:::::::::
      4 = π::BBBB♦::
      5 = π:::::::::
      6 = π::::::CCC
      7 = π:::::::::
      8 = DDDD♦:EEEE
      9 = π:::::::::
     10 = *♦FFFF♦:::
     11 = π:::::::::
     12 = π:::GGGG♦
     13 = π:::::::::
     14 = IIII♦:::::
     15 = π:::::::::
     16 = JJJKKKK♦::
     17 = π:::::::::
```

```
:: MEHET =I/N=? N
```

Elemezzük az eredményeket!

– A 2. rekordban:
Ha egy adatot felviszünk egy rekordba, akkor a PRINT utasítás végén lévő RETURN jel is felkerül, ugyanúgy mint a soros vagy random állományoknál. Az adat és e jel által le nem fedett üres bájtok érintetlenül maradnak.
Megjegyezzük, hogy a RETURN jel felvitele a PRINT végén kiadott pontosvesszővel letiltható.

– A 4. rekordban:
Ha egy adatot a rekord belsejébe pozicionálva írunk fel, az előtte lévő bájtok érintetlenül maradnak.

– A 6. rekordban:
Ha egy adat nem fér be a rekordba, akkor annyi karakter kerül belőle oda, amennyi befér, utána **OVERFLOW IN RECORD** (rekordhatár túllépése) hibaüzenetet kapunk.

— A 8. rekordban:

Ugyancsak **OVERFLOW IN RECORD** hibaüzenetet kapunk, ha maga az adat elfér ugyan a rekordban, de az utána következő RETURN jel már nem.

Ha a rekordba kerülő adatokat nem hézagmentesen pozicionáljuk, akkor köztük megmaradnak az üres (bináris nulla) bájtok.

— A 10. rekordban:

Ha egy részben feltöltött rekordba viszünk fel adatot, az újonnan felvitt adat előtt már fenn lévő adatok érintetlenül maradnak.

Ha minden adatot külön menetben viszünk fel, szigorúan balról jobbra haladva a feltöltéssel, akkor megfelelő pozicionálással elérhető, hogy az adatok között szeparátorjelként a RETURN jel szerepeljen. (Ez a későbbi olvasáskor kihasználható lesz.)

— A 12. rekordban:

Ezt a rekordot csak azért vittük fel, hogy összehasonlíthassuk a 14. rekorddal; bizonyítva, hogy ott az ugyanígy felvitt "HHHH" adat nem azért hiányzik, mert nem sikerült felírni.

— A 14. rekordban:

Ha egy részben feltöltött rekordba a fent lévő adatok elé pozicionálva viszünk fel egy újabb adatot, akkor az újat követő régi adatok megsemmisülnek. (Emlékezzünk arra, hogy a 14. rekordba először a "HHHH" adatot vittük fel a 6-os bájtpozíciótól kezdve, ugyanúgy, mint a 12. rekord esetén a "GGGG" adatot. Ezután került az "IIII" adat a 14. rekord 1. pozíciójára. Minderről ismételten meggyőződhetünk, ha a "HHHH" és "IIII" adatokat ugyanilyen módon felírjuk például a 18. rekordba, de minden egyes írás után a RELELLENORZES programmal megnézzük a rekord tartalmát.)

A megsemmisülés akkor is bekövetkezik, ha az adatok között nincs átfedés.

— A 16. rekordban:

Ha egy részben feltöltött rekordba új adatot viszünk fel úgy, hogy a kezdőpozíciója „belelóg" egy már fent lévő adatba, akkor annak az új adattal lefedett bájtjai felülíródnak. (A rekordban eredetileg "JJJJ" adat volt, amit még egy RETURN jel is követett.)

A korábbi tapasztalatoknak megfelelően az új adat előtti bájtok érintetlenül maradnak, az azt követőek pedig megsemmisülnek.

Most pedig állítsuk le a programot a "MEHET?" kérdésre adott tagadó válasszal, és foglaljuk össze tömören a tapasztalatokat.

A relatív állomány rekordjaiba tetszőleges pozíciókra vihetjük fel az adatainkat, ügyelve arra, hogy:

- az adatok a rekordban elférjenek;
- az adatok közötti RETURN jelek számára is legyen elegendő hely;
- az adatok ne fedjék át egymást;
- a rekord feltöltése szigorúan balról jobbra haladjon.

Természetesen a RETURN jelre csak akkor van szükségünk, ha olvasáskor ezt használni akarjuk. A balról jobbra haladó feltöltés sem kötelező, ha gondoskodunk az új adattól jobbra lévő régi adatoknak az új felírása előtti kimentéséről, majd a felírás utáni vissza-

töltéséről; hacsak nem éppen a régi adatok megsemmisítése vagy a rekordszerkezet átalakítása a célunk.

Végül még egy megjegyzés az írással kapcsolatban. Töltsük be, és indítsuk el a RELIRAS programot az alábbi paraméterekkel:

```
RELATIV ALLOMANYIRASA

? ALLOMANY  NEVE =? MZ/X
? REKORDSORSZAM  =? 10
? BAJTPOZICIO    =? 1
? FELIRANDO ADAT =? AKARMI

62 : FILE NOT FOUND
SOR= 220            OPEN
```

Már a megnyitáskor **FILE NOT FOUND,** vagyis az állomány nincs a lemezen hibaüzenetet kapunk, mert ilyen nevű állományt még valóban nem hoztunk létre.

Írni tehát csak már létező állományba lehet, kivéve, ha az állományt létrehozásra nyitjuk meg, ahogyan azt a RELGENERALAS programban tettük. Ilyenkor természetesen írhatunk is bele, éspedig nemcsak egyetlen rekordot, hanem akárhányat.

FIGYELEM!
Sem a RELGENERALAS, sem a RELIRAS program nem a relatív állomány feltöltésére való, hanem csupán az üres állomány létrehozására, illetve a rekordok feltöltésének, vagyis a rekordkezelésnek a kipróbálására. Az állományba általában nem így kerülnek be az adatok, hanem úgy, ahogyan majd a karbantartásnál látni fogjuk.

FIGYELEM!
Ahogy a „SOROS LEMEZÁLLOMÁNYOK" című kötetben a rekordkezeléssel összefüggésben már részletesen tárgyaltuk, a rekordon belüli adatok között a szeparátorjel vessző, azaz CHRS(44) legyen, és csak a rekord végén álljon RETURN, azaz CHRS(13) jel — ha a programban a relatív állomány szellemének megfelelően valóban rekordorientált feldolgozást akarunk végrehajtani. (A RELIRAS programban csak azért használtunk mindenütt a sorvége (RETURN) jelet, mert itt szó sincs rekordonkénti feldolgozásról, csupán az írás teszteléséről — ami így egyszerűbb.)

A RELATÍV ÁLLOMÁNY KIBŐVÍTÉSE

Mint már tapasztaltuk, a "PROBA" állományban 25 rekord van. Mit tehetünk, ha mondjuk a 33. rekordba akarunk egy adatot felvinni?

Próbáljuk ki! A már betöltött RELIRAS programot indítsuk el úgy, hogy az alábbi paramétereket adjuk meg:

```
RELATIV ALLOMANY IRASA
```

```
:: ALLOMANY  NEVE =? PROBA
:: REKORDSORSZAM  =? 33
:: BAJTPOZICIO    =? 1
:: FELIRANDO ADAT =? BOVITES
```

```
FELIRVA
```

A program simán le fog futni, és FELIRVA üzenetet ad. Ekkor töltsük be a RELELLENORZES programot, és futtassuk le a "PROBA" állomány 21–40 rekordjaira, rekordonként 10 bájtot olvasva.

```
21 = π: : : : : : : : :
22 = π: : : : : : : : :
23 = π: : : : : : : : :
24 = π: : : : : : : : :
25 = π: : : : : : : : :
26 = π: : : : : : : : :
27 = π: : : : : : : : :
28 = π: : : : : : : : :
29 = π: : : : : : : : :
30 = π: : : : : : : : :
31 = π: : : : : : : : :
32 = π: : : : : : : : :
33 = BOVITES●: :
34 = π: : : : : : : : :
35 = π: : : : : : : : :
36 = π: : : : : : : : :
37 = π: : : : : : : : :
38 = π: : : : : : : : :
39 = π: : : : : : : : :
40 = π: : : : : : : : :
```

```
:: MEHET =I/N=? I
```

A képernyőn ott fog díszelegni a 33-as rekord a "BOVITES" szöveggel feltöltve. Az állomány egyébként a 26. rekordtól az 50.-ig üres rekordokkal lesz feltöltve. Erről meggyőződhetünk, ha a lekérdezést tovább folytatjuk a 41–60 intervallummal.

Az íráskor tehát a lemezkezelő automatikusan kibővítette az állományunkat egy blokkal.

Természetesen ha van hely a lemezen, akár több blokkal is bővíteni tudja az állományt, például ha az 555-ös rekordba akarnánk adatot felvinni.

Ha a bővítés ilyen egyszerű, miért kell a létrehozáskor lefoglalni az állomány számára a lemezterületet? Nem lenne jobb, ha üres állományt hoznánk létre, vagy legalábbis olyan kicsit, amilyet csak lehet, és azután szükség szerint bővítenénk?

Az ötlet elvileg jó, a megvalósításának nincs akadálya. Próbáljuk ki! Töltsük be a RELGENERALAS programot, és töröljük ki belőle a 240–280 sorokat, hogy az OPEN és a CLOSE között csak egy hibavizsgálat maradjon:

```
200 RH=INT(S/256)
201 RL=S-RH*256
210 OPEN 15,8,15
220 OPEN 2,8,2,F$+",L,"+CHR$(R)
230 L=220: L$="OPEN ": GOSUB 1010
240 REM -----TOROLVE-----
250 REM -----TOROLVE-----
270 REM -----TOROLVE-----
280 REM -----TOROLVE-----
310 CLOSE 2
320 CLOSE 15
```

Indítsuk el! Az állomány neve legyen "MINTA", a rekordméret 10 bájt, a rekordok száma 10, a végjel csillag.

A program normálisan le fog futni. Bejegyzi az állomány nevét a tartalomjegyzékbe, és megjegyzi a rekordméretet is. De ha bekérjük a lemez tartalomjegyzékét, látni fogjuk, hogy a "MINTA" nevű REL állomány által elfoglalt blokkok száma nulla.

Most töltsük be a RELIRAS programot, és futtassuk le kétszer:

```
▒ ALLOMANY  NEVE =? MINTA
▒ REKORDSORSZAM  =? 10
▒ BAJTPOZICIO    =? 1
▒ FELIRANDO ADAT =? *

▒ ALLOMANY  NEVE =? MINTA
▒ REKORDSORSZAM  =? 10
▒ BAJTPOZICIO    =? 3
▒ FELIRANDO ADAT =? MINTA
```

Ezután ellenőrizzük a "MINTA" állományt a RELELLENORZES programmal, rekordonként 10 bájtot olvasva. Látni fogjuk, hogy megvan mind a 25 rekord, amelyek közül 24 üres, míg a 10.-ben ott a csillag és a "MINTA" szöveg.

A módszer tehát működik. Az egyetlen kifogás ellene az, hogy a kiterjesztés lassítja a feldolgozást, mivel az nagyon időigényes; ugyanis a lemezkezelőnek minden egyes alkalommal teljes blokkot vagy blokkokat kell teleírnia.

Helyesebb tehát az állomány méretét a lehető legpontosabban megbecsülni, és a megfelelő méretű lemezterületet a létrehozáskor lefoglalni.
Ilyenkor a lemezterület időigényes előkészítését csak egyetlen egyszer kell „végigszenvednünk", mivel azonban a kiterjesztésre csak kivételesen kerül sor, az állomány kezelése valamivel gyorsabb lesz.
Mellesleg így nem érhet bennünket az a meglepetés, hogy mondjuk az állomány felének tényleges feltöltése után derül ki, hogy további rekordok számára a lemezen már nincs hely.

A RELATÍV ÁLLOMÁNY TÖRLÉSE

A relatív állomány kizárólag a parancscsatornára kiadott SCRATCH lemezkezelő funkcióval törölhető.
Például:

```
OPEN 15,8,15

READY.
PRINT#15,"SCRATCH:PROBA"

READY.
CLOSE 15
```

FIGYELEM!
A relatív állomány újbóli létrehozása nem jelenti az állomány törlését.

Ha egy létező relatív állományt létrehozásra nyitunk meg, akkor ennek hatása olyan lesz, mintha írásra nyitottuk volna meg:

- Ha nem adunk ki PRINT utasítást, az állomány az eredeti állapotában marad.
- Ha létező rekordra adunk ki PRINT utasítást, az adatot felviszi a kérdéses rekordba.
- Ha nemlétező rekordra adunk ki PRINT utasítást, akkor a lemezkezelő a megfelelő mértékben kiterjeszti az állományt, és feltölti a kérdéses rekordot.

Mindezt a RELGENERALAS programmal kipróbálhatjuk, akár a "MINTA", akár (ha még nem töröltük a "PROBA" állományra.

Rekordméretként mindig az eredeti rekordméretet, a 10-et, végjelként mindid a csillagtól feltűnően eltérő "%" jelet adjuk meg. Az eredmény a RELELLENORZES programmal vizsgálható meg.

FIGYELEM!
Ha a létrehozott állományt az eredetitől eltérő rekordmérettel kíséreljük meg újra létrehozni, a legelső pozicionálás alkalmával **NO CHANNEL,** vagyis az állomány számára „nincs hozzáférhető szabad adatcsatorna" hibaüzenetet kapunk, a létrehozás pedig sikertelen lesz.

Ez a hiba leginkább olyankor fordul elő, amikor egy új állományt akarunk létrehozni, de elfeledkezünk arról, hogy a megadott néven már van egy állományunk a lemezen; vagy ha az állomány nevét hibásan gépeljük, és ezzel egy létező névbe „találunk bele".

A hibaüzenet kissé meglepő, de a RELGENERALAS program lefuttatásával meggyőződhetünk róla, hogy valóban ezt az üzenetet kapjuk, amikor például a "PROBA" vagy a "MINTA" állomány újbóli létrehozásakor bármilyen, 10-től eltérő rekordméretet adunk meg.

REKORD TÖRLÉSE A RELATÍV ÁLLOMÁNYBÓL

Az állomány szerkezeti felépítéséből és szervezési módjából következik, hogy rekordot fizikailag törölni, vagyis az állományból eltávolítani lehetetlen.
A rekord tartalma azonban törölhető. Ennek az a legegyszerűbb módja, hogy üres rekorddá alakítjuk, azaz hogy az első bájtra visszaírjuk a CHR$(255) karaktert, vagyis a π jelet, éspedig úgy, hogy utána ne következzen RETURN jel.
A RELELLENORZES programmal nézzük meg, hogy mi van a "MINTA" állomány 10. rekordjában. Ha a könyvben leírtak szerint jártunk el, akkor ez:

```
 9 = π:::::::::
10 = *♦MINTA♦::
11 = π:::::::::

MEHET =I/N=? N
```

Ezután töltsük be a RELIRAS programot, és tegyünk a 270-es sorban lévő PRINT utasítás A$ változója után egy pontosvesszőt. Majd indítsuk el a programot:

```
ALLOMANY  NEVE =? MINTA
REKORDSORSZAM  =? 10
BAJTPOZICIO    =? 1
FELIRANDO ADAT =? π

 9 = π:::::::::
10 = π:::::::::
11 = π:::::::::

MEHET =I/N=? N
```

Töltsük be a RELELLENORZES-t, és nézzük meg a rekordot; az ugyanolyan lesz, mint a többi üres rekord.
Az ilyen törléskor kihasználjuk, hogy egy adat felvitele megsemmisíti a rekordban tőle jobbra lévő régi adatokat. A törlést tehát felülírással valósítjuk meg.
Igen gyakori azonban, hogy nem ezt a módszert választjuk, hanem a rekord első bájtját egy állapotjelzőnek tekintjük, ahová mindig egy, a rekord állapotára jellemző jelet viszünk fel. Például feltöltött rekord esetén "+", törölt rekord esetén "–" jelet. Az üres rekordban nyilván meghagyjuk a π jelét. Ilyenkor további jelek kiválasztásával megkü-

lönböztethetjük egymástól a teljesen, illetve a részben feltöltött rekordokat, vagy akár – mint ahogyan a létrehozáskor tettük – a csillaggal az állomány fizikai végét a logikai végétől. (Ezek a megkülönböztető jelek a karbantartáskor előnyösen kihasználhatók.)

A törölt rekord ilyesfajta megjelölésének csak akkor van értelme, ha a "–" jel beírásakor nem töröljük a rekordtartalmat, hanem visszaírjuk a rekordba. Ugyanis ilyenkor a törölt rekord eredeti állapota egyszerűen helyreállítható.

Ott, ahol az adatvesztés komoly károkat okozhat, és ezért nagy biztonságra kell törekednünk (azaz a téves törléseknek helyreigazíthatóknak kell lenniük), vagy ott, ahol a rekordok ideiglenes törlése is előfordulhat, célszerű ezt a megoldást választanunk. Persze nem ez az egyetlen lehetőség. A törölt rekordokat például kimenthetjük egy tartalék állományba is; ilyenkor a helyük a törzsállományban felszabadul, viszont a rekord helyreállítása valamivel körülményesebb.

A relatív állományok karbantartásával kapcsolatban mindkét módszerre látunk majd példát.

A RELATÍV ÁLLOMÁNY OLVASÁSA

Általában nem a rekord bájtjainak tartalmára, hanem a rekordban tárolt adatokra van szükségünk. Így az ellenőrzés során megvalósított bájtonkénti olvasás helyett gyakrabban olvasunk adatokat.

Az adatbeolvasás lényege nem új. Mindig két lépésből áll:

– POZICIONÁLÁS: először be kell állítanunk azt a rekord- és bájtpozíciót, amelyről az olvasásnak el kell kezdődnie.
– OLVASÁS: az adat tényleges beolvasása, amely a pozicionálás után hajtható végre.

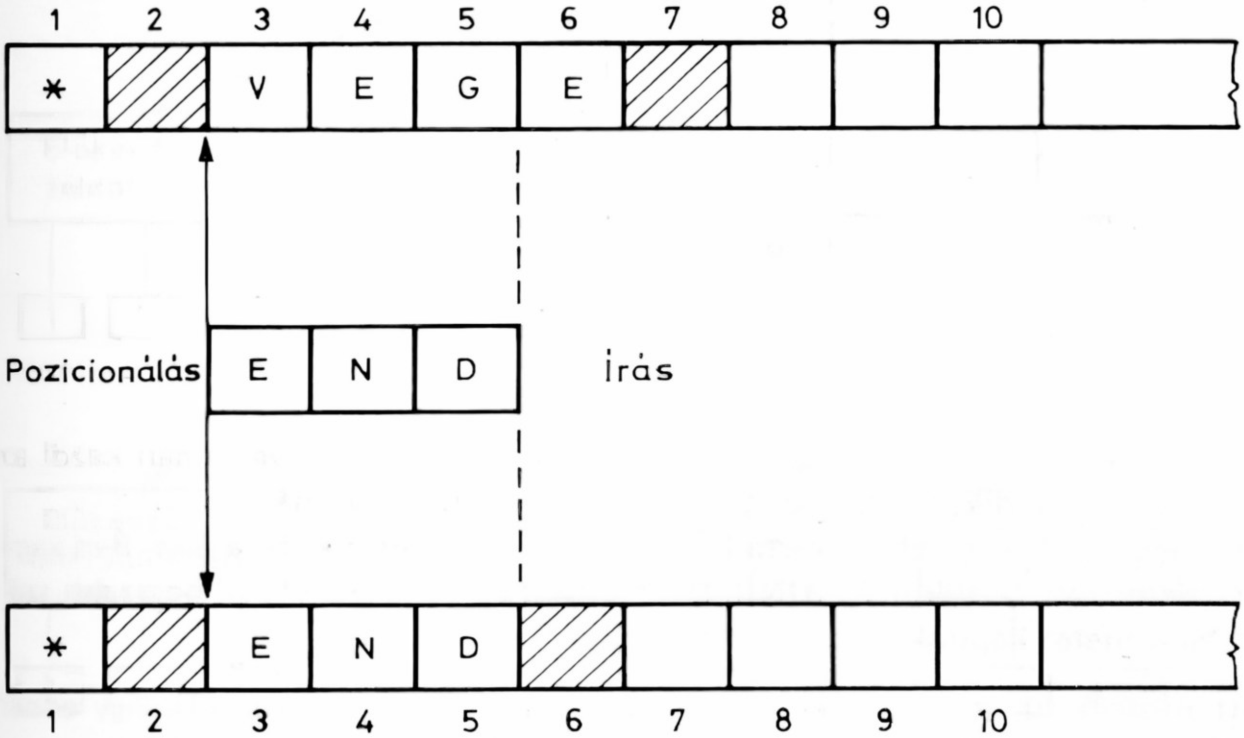
Mindkettő a már ismert módon történik; a pozicionálás a parancscsatornára PRINT utasítással kiírt pozicionáló információkkal, az olvasás pedig az állományra kiadott INPUT utasítással.

A rekordra már jól tudunk pozicionálni: a rekordsorszámot (RS) konvertáljuk egy alsó (RL) és felső (RH) bájtból álló bináris formára, és ezeket használjuk pozicionáló információként. A pozicionálás sikerességéről a parancscsatorna H hibakódjának megvizsgálásával győződhetünk meg: H=0 esetén a pozicionálás sikeres volt, minden más esetben nem. H=50 esetén a sikertelenség oka, hogy a rekord nincs az állományban. Minderre elegendő példát láttunk.

Most tehát csak a bájtokra való pozicionálás lehetőségeinek és hatásának elemzésével foglalkozunk. Ehhez azonban tudomásul kell vennünk, hogy a rekordkezelés érdekében az adatok közé, valamint az utolsó adat végére ugyanolyan szeparátorjeleket kell elhelyeznünk, mint amilyeneket a soros és a random állományoknál is használtunk; azaz vesszőt vagy a RETURN jelet. Ezek a jelek természetesen a rekordméretbe beszámítanak. (A rekordkezeléssel a „SOROS LEMEZÁLLOMÁNYOK" című kötetben részletesen foglalkoztunk.)

A pozicionálásnak már az íráskor is lényeges szerepe van, és nemcsak annyi, hogy ott is meg kell adnunk, hogy a rekord feltöltése melyik bájttól kezdődjön; mint ezt a RELIRAS tesztprogramban is tettük.

Vegyünk például egy olyan rekordot, amelynek az első bájtján egy állapotjelző van, a harmadikon pedig a "VEGE" szöveg kezdődik. Az adatokat RETURN jel követi. Ha most a 3-as bájtra pozicionálva az "END" szöveg felírására adunk ki utasítást, a rekord tartalma így módosul (27. ábra):



27. ábra. Relatív írás

Vagyis a "VEGE" helyett nem "ENDE", hanem "END" szöveg szerepel a rekordban, amelynek a feltöltött része egy bájttal rövidebb lesz.

Az írás nagyon lényeges tulajdonsága, hogy soha nem bájtonként, hanem mindig adatonként történik, úgy hogy a rekordnak a megadott pozíció előtti része érintetlen marad, a pozíciótól számított, az adat méretének megfelelő része pedig felülíródik a megadott adattal, míg a rekord fennmaradó része törlődik, azaz oda bináris nullák kerülnek.

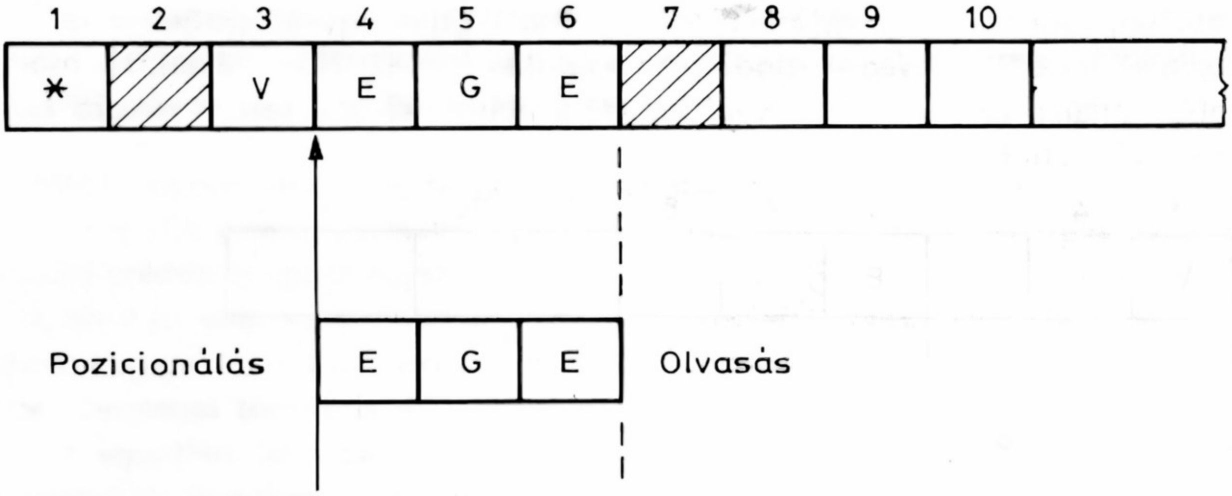
Mindezt már kipróbáltuk, de ha kívánjuk, a fenti példa kapcsán ismét meggyőződhetünk róla. Vigyük fel a rekordot a RELIRAS programmal például a "MINTA" állomány 10-es rekordpozíciójára. Nézzük meg a RELELLENORZES programmal, majd a példának megfelelően módosítsuk a rekordot a RELIRAS programmal, és vizsgáljuk meg az eredményt ismét a RELELLENORZES programmal.

Olvasáskor is ugyanez az adatkezelési szemlélet érvényesül. Mindig pozicionálás előzi meg, azaz meg kell adnunk, hogy az olvasás a rekord melyik bájtjától kezdődjön. Ezután akár egy adatot olvasunk be, akár többet, minden egyes adat olvasása a legközelebbi szeparátorjelig tart.

Így ha a 28. ábra szerinti

rekordból egy R$ adat beolvasására adunk utasítást:

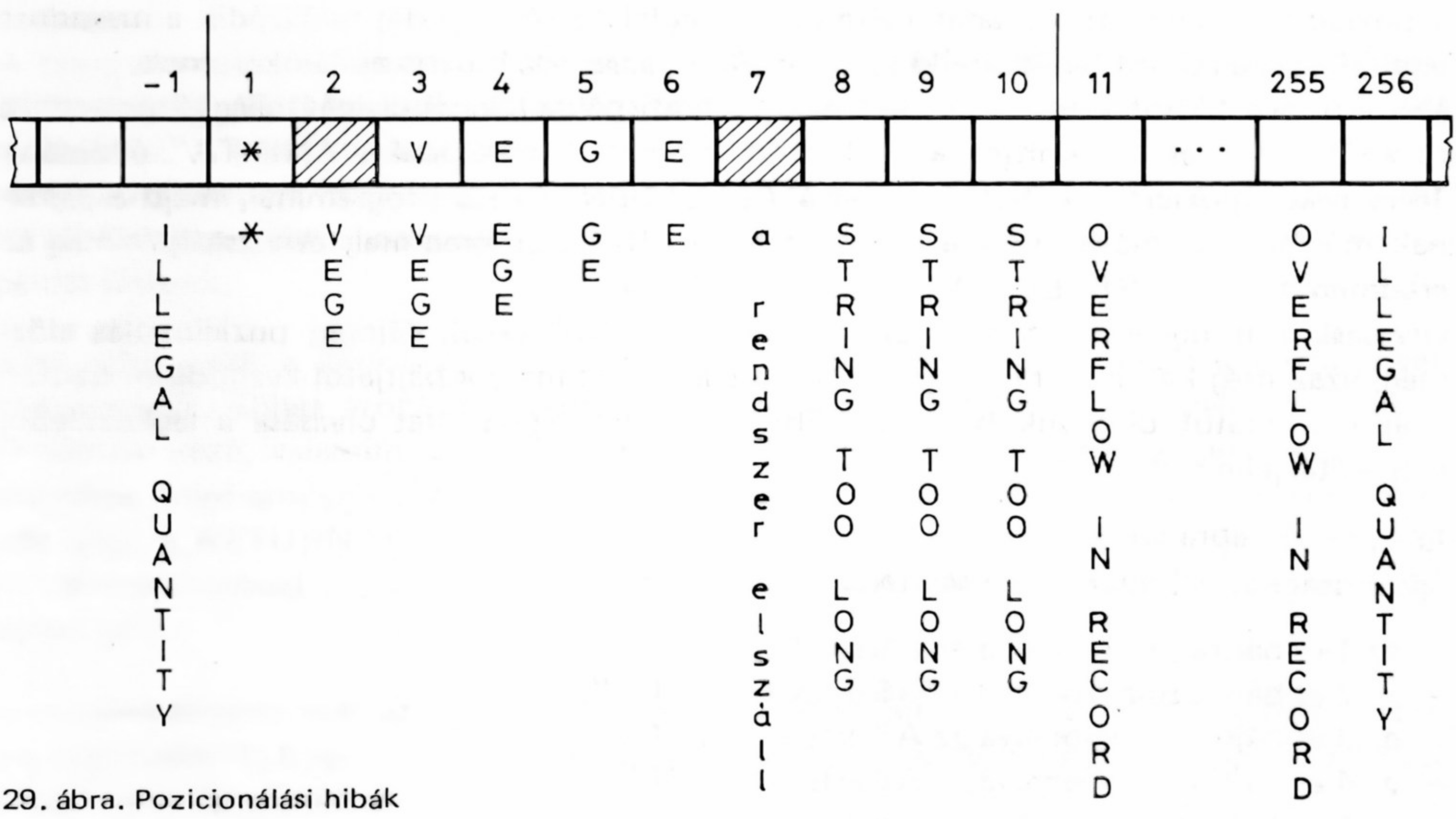
– az 1-es bájtra pozicionálva az A$ értéke = "*"
– a 2-es bájtra pozicionálva az A$ értéke = "VEGE"
– a 3-as bájtra pozicionálva az A$ értéke = "VEGE"
– a 4-es bájtra pozicionálva az A$ értéke = "EGE"
– az 5-ös bájtra pozicionálva az A$ értéke = "GE"
– a 6-os bájtra pozicionálva az A$ értéke = "E"



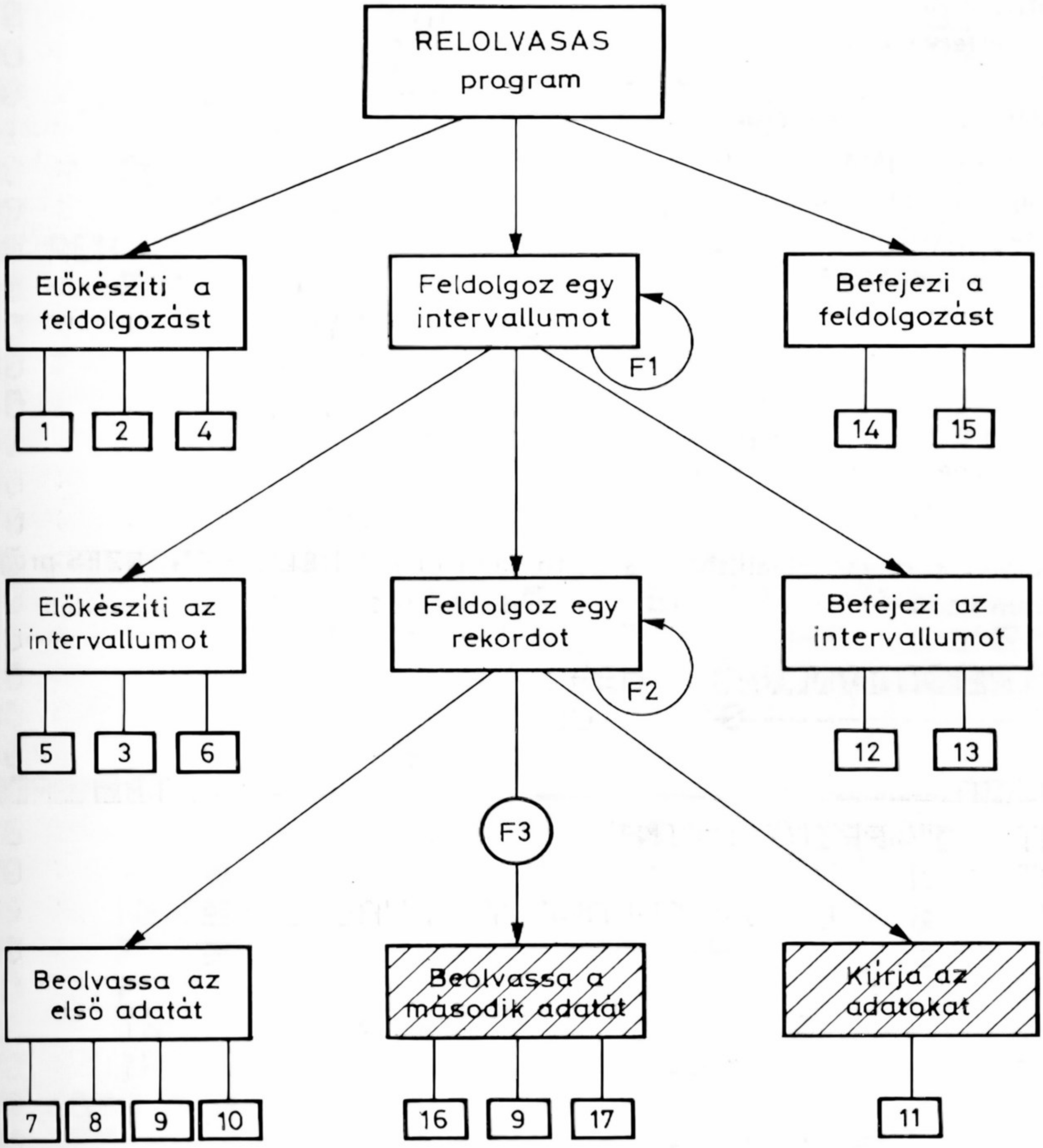
28. ábra. Relatív olvasás

Ha a 0-s bájtra pozicionálunk, a lemezkezelő a rekord elejére ír, illetve onnan kezdi az olvasást. Vagyis úgy viselkedik, mintha az 1. bájtra pozicionáltunk volna.
Ha a rekord fel nem töltött részéből akarunk olvasni, azaz az esetünkben a 8-as, 9-es vagy 10-es bájtra pozicionálunk, akkor **STRING TOO LONG** („megengedettnél hosszabb szöveges adat") hibaüzenetet kapunk.

Ha a beállított pozíció túlmutat a rekordterületen, azaz ha példánkban 11 vagy ennél nagyobb, **OVERFLOW IN RECORD** („rekordhatár túllépése") hiba következik be.
Negatív vagy 255-nél nagyobb pozíció beállítása **ILLEGAL QUANTITY** („nem megengedett paraméterérték") hibaüzenettel programmegszakítást eredményez.
Ha pedig a rekord végjelére pozicionálunk, azaz példánkban a 7. bájtra, akkor a rendszer hibaállapotba kerül, és csak a STOP RUN és a RESTORE gomb együttes megnyomásával hozható ki onnan. Összefoglalásként nézzük a 29. ábrát!



29. ábra. Pozicionálási hibák



30. ábra. Relatív állomány olvasása

Mindezek kipróbálására külön programot írhatunk, amely a bájtonkénti olvasás (RELELLENORZES) mintájára adatonként olvas (30. ábra).

A feltételek:

– F1: ha mehet a következő intervallum feldolgozása, azaz ha V$="I";
– F2: ha az aktuális RS rekordsorszám nem kisebb az intervallum A alsó határánál, és nem nagyobb az intervallum F felső határánál;
– F3: ha a rekord nem üres, azaz ha az első A$ bájt értéke nem=CHR$(255).

Vessük össze ezt a bájtonként olvasó (RELELLENORZES) programmal. Látható, hogy szerkezeti változás csak a legalacsonyabb szinten fordul elő. Valamivel több eltérés van a (∗-gal megjelölt) résztevékenységekben, ahol is a program:

– 1: bejelentkezik;
– 2: bekéri az állomány nevét;
∗ 3: bekéri a bájtpozíciót;

— 4: megnyitja az állományt;
— 5: bekéri az intervallumot;
— 6: fejlécet készít;
— 7: konvertálja a rekordsorszámot;
* 8: a rekord első bájtjára pozicionál;
— 9: hibavizsgálatot hajt végre;
* 10: beolvassa a jelzőbájtot;
* 11: kiírja az adat tartalmát;
— 12: láblécet készít;
— 13: megkérdezi, jöhet-e a következő intervallum;
— 14: kijelentkezik;
— 15: lezárja az állományt;
* 16: a rekord megadott bájtjára pozicionál;
* 17: beolvassa az adatot.

Ennek megfelelően könnyen előállítható a bájtonként olvasó RELELLENORZES programból a program kódja:

```
1 REM:     RELATIVOLVAS
2 REM:     ------------
3 REM:
100 REM -----------------------------------------ADATBEKERES
101 PRINT "□":PRINT:PRINT
102 PRINT "  ■                              ■"
103 PRINT "  ■ RELATIV ALLOMANY OLVASASA ■"
104 PRINT "  ■                              ■"
105 PRINT:PRINT
110 PRINT
120 INPUT "■*■ ALLOMANY=";F$
130 L=LEN(F$)
140 :   IF 1>L OR L>16 THEN GOTO 120
150 PRINT:PRINT
160 PRINT
170 INPUT "■*■ BAJTPOZICIO=   3■■■";P
180 P=INT(P)
190 :   IF P<0 OR P>253 THEN GOTO 160
200 REM ------------------------------------------MEGNYITAS
210 OPEN 15,8,15
220 OPEN 2,8,2,F$
300 REM ---------------------------INTERVALLUMBEKERES
310 PRINT "□": PRINT
320 PRINT "■*■ INTERVALLUM "
330 :   PRINT
340 :   INPUT "  REKORDTOL =   0■■■";A
350 :   PRINT
360 :   INPUT "  REKORDIG  =   0■■■";F
```

```
370 :  :   IF A=0 AND F=0      THEN GOTO 810
380 :  :   IF A<1 OR  A>65535 THEN GOTO 330
390 :  :   IF F<A OR  F>65535 THEN GOTO 330
391 :  :   IF F-A > 20         THEN GOTO 330
398 :  PRINT "▟"
399 :  PRINT " ─────────┬─────┬┬┬┬┬┬┬┬┬┬┬┬┬┬┐"
400 REM -----------------------------------ADATOLVASAS
410 :  FOR RS=A TO F
420 :  :  RH=INT(RS/256)
430 :  :  RL=RS-RH*256
440 :  :  PRINT#15,"P"CHR$(2)CHR$(RL)CHR$(RH)CHR$(1)
450 :  :  GOSUB 1010 :REM==> HIBAVIZSGALAT
460 :  :  IF H>=20 THEN GOTO 710
470 :  :  :  INPUT#2,A$
475 :  :  :  B$=""
480 :  :  IF A$=CHR$(255) THEN GOTO 610
510 :  :  :  PRINT#15,"P"CHR$(2)CHR$(RL)CHR$(RH)CHR$(P)
520 :  :  :  GOSUB 1010 :REM==> HIBAVIZSGALAT
530 :  :  IF H>=20 THEN GOTO 710
540 :  :  :  INPUT#2,B$
600 REM ------------------------------------ADATKIIRAS
610 :  :  R$=RIGHT$("     "+STR$(RS),6)
620 :  :  PRINT R$;" : ";A$;" : ";B$
699 :  NEXT RS
700 :  PRINT " ─────────┴─────┴┴┴┴┴┴┴┴┴┴┴┴┴┴┘"
710 :  PRINT
720 :  INPUT "▒*█ MEHET =I/N=  I███";V$
730 :  IF V$<>"I" THEN GOTO 810
799 GOTO 310
800 REM ----------------------------------KIJELENTKEZES
810 PRINT:PRINT"▒*█ VEGE"
900 REM ----------------------------------------LEZARAS
910 CLOSE 2
920 CLOSE 15
999 END
1000 REM ========================HIBAVIZSGALAT
1010 INPUT#15,H,H$,S,B
1020 IF H<20 THEN GOTO 1999
1030 :  PRINT " ─────────┴─────┴┴┴┴┴┴┴┴┴┴┴┴┴┴┘"
1040 :  PRINT " ▒";RS;" - ";H$;"! █"
1999 RETURN
```

A 102–104, 399, 700, 1030–1040 sorokban csak „kozmetikázást'' hajtottunk végre; az üzeneteket és a fej-, illetve láblécet igazítottuk az adatonkénti olvasáshoz.

A 392–396 sorokban lényegileg csak az az új, hogy a bájtpozíció bekérése új helyre került. Így minden egyes intervallumban más-más pozícióról olvashatunk, illetve ugyanazt

az intervallumot többször egymás után választva, a rekordnak akár az összes adatát beolvashatjuk.

A bájtpozíciót nem érdemes 3-nál kisebbre választanunk, hiszen a próbaállományainkban az előtt nem adat, hanem az állapotjelző és annak szeparátorjele van. Persze megadhatunk 0 vagy 1 pozíciót is, de ekkor adatként az állapotjelzőt fogjuk beolvasni.

Értelemszerűen a legnagyobb eltéréssel a tulajdonképpeni feldolgozást végző modulban, a 440–540 sorok között találkozunk.

Az természetes, hogy a GET helyett INPUT utasítást használunk. Az is, hogy itt nincs szükségünk a bájtonkénti olvasás ciklusára, hiszen akárhány bájtból áll az adat, egyetlen utasítással olvasható be.

Feltűnő, hogy az olvasás látszólag két részben történik. A 440–475 sorokban először az A$ állapotjelzőt olvassuk be az első bájtpozícióról. Maga a beolvasandó adat még üres. Erre csak azért van így szükség, hogy ne olvassunk bele üres rekordba. Ez az INPUT utasítás tehát funkcionálisan nem olvasás, csak a rekord ellenőrzése. Voltaképpen GET utasítást is használhattunk volna, hiszen az állapotjelző egyetlen jelzőbájtból áll.

Az adat tényleges beolvasása csak akkor következik be, ha az állapotjelző feltöltött rekordot jelez, azaz ha a tartalma nem CHR$(255). Ezt figyeli a 480-as sor.

Az adatot az 510–540 sorokban olvassuk be, az előzőleg megadott P bájtpozícióról a B$ változóba.

A kiírás a 620-as sorban ennek megfelelően három információt tartalmaz: az RS rekordsorszámot, az A$ állapotjelzőt, majd a P pozícióról beolvasott B$ adatot.

Ha a programot előállítottuk a bájtonként olvasó programból, mentsük ki, mondjuk RELOLVASAS néven. Majd próbáljuk ki: Például:

```
RELATIV ALLOMANY OLVASASA

:] ALLOMANY=? PROBA

:] BAJTPOZICIO=? 3

 9 : π :
10 : * : FFFF
11 : π :

:] MEHET =I/N=? I
```

A 3-as bájtra pozicionáltunk, ezért minden feltöltött rekordból csak az első adatot kapjuk meg (az állapotjelzőn kívül). Az üres rekordok esetén maga az adat is üres.

Próbálgassuk a programot különböző bájtpozíciókkal, és vessük össze a tapasztalatainkat a korábban közölt táblázattal. A 396-os sorban a program figyeli a pozíció formális helyességét. Ha a negatív vagy a 253 érték fölötti bájtpozicionálás hatását is látni akarjuk, a figyelést ki kell kapcsolni, például a 396-os sor törlésével.

A programot gondosan őrizzük meg, mert bármilyen relatív állományból tud adatokat olvasni. Erre teszteléskor még szükségünk lehet.
Megjegyezzük, hogy — mint a programból is látható — üres rekordban kizárólag az első bájtra biztonságos pozicionálni. Ez persze rekordorientált feldolgozáskor nem jelent megkötést, hiszen ott amúgy is ezt tesszük. Ilyenkor a rekord első változója a CHR$(255) értéket kapja, a többi pedig 0-t vagy " "-t, azaz nullát vagy üres karaktert, attól függően, hogy numerikus vagy szöveges változóról van-e szó.

Ha viszont nem teljes rekordot, hanem abból csak meghatározott adatot vagy adatsorozatot akarunk olvasni, vagyis nem az első bájtra pozicionálunk, számolnunk kell a fentebb ismertetett hibákkal. Erről könnyen meggyőződhetünk, ha betöltjük a RELOLVASAS programot, és a 345 GOTO 475 utasítással kikapcsoljuk benne az állapotjelző figyelését. Indítsuk most el a "PROBA" állományra, az első 10 rekordot olvasva a 3. bájttól. Az első és a harmadik rekordból "FFFF" olvasódik be, noha mindkét rekord üres. A másodikból a várakozásnak megfelelően "AA" jön be; majd a program a negyedik rekordnál elszáll, és csak STOP + RESTORE hatására jön vissza a rendszer. Ha ezután a programot újraindítjuk, de az első bájttól olvastatunk vele, rendben lefut.

Az olvasás több adattal is kipróbálható: az 540 INPUT utasítást egészítsük ki a ,C$,D változókkal, a 620 PRINT utasítást pedig bővítsük a ;"!";C$;"!";D;"!"adatokkal. Futtassuk a programot a "PROBA" állomány első 10 rekordjára, a 3. bájttól olvasva. Már az első rekordnál programmegszakítás következik be **FILE DATA ERROR** hibaüzenettel. Ha újraindítjuk a programot úgy, hogy az olvasást az első bájttól kezdje, rendben le fog futni, és látható lesz a fel nem töltött változók " ", illetve 0 értéke, amit akkor is felvesznek, ha olvasás előtt C$="X", illetve D=9 kezdőértéket kapnak.

A RELATÍV ÁLLOMÁNY KEZELÉSE

A relatív állomány kezelése tágabb értelemben a létrehozást, karbantartást és feldolgozást foglalja magában.
A létrehozással már megismerkedtünk.
A karbantartás tulajdonképpen az állomány egyes rekordjainak, pontosabban ezek tartalmának megváltoztatását jelenti. Ennek megfelelően az alábbi funkciókról beszélhetünk:

– FELVITEL,
melynek során új adatokat viszünk fel egy még feltöltetlen rekordba. (Ha a szóban forgó rekord nem szerepel az állományban, ez a művelet automatikusan az állomány kiterjesztésével jár együtt.)

– MÓDOSÍTÁS,
melynek során egy meglévő rekord adatait cseréljük ki új adatokkal. (A módosítás lehet teljes, amikor a rekord összes adatát kicseréljük, de lehet részleges is, amikor csak meghatározott adatokkal tesszük ezt.)

– TÖRLÉS,
melynek során egy meglévő rekord adatait töröljük. (Emlékezzünk arra, hogy csak a rekord adatai törölhetők. Maga a rekord sajátos fizikai szerkezete miatt nem távolítható el az állományból.

Elvileg mindhárom funkció ugyanarra a műveletre, az írásra épül: először pozicionálunk a karbantartandó rekordra, majd felírjuk az adatokat a rekordterületre. Ha a rekordot új adatokkal töltöttük fel, akkor tulajdonképpen felvitelt hajtottunk végre; ha meglévő adatok módosított változataival tettük ugyanezt, akkor módosítást végeztünk; míg ha a rekordot egyezményes jelekkel írtuk tele, voltaképpen a rekord adatainak a törlését valósítottuk meg.

Vegyük észre, hogy itt végül is mindig egy már létező — vagy bővítéskor egy frissen létrehozott — rekord felülírása történik meg. A felvitel tehát megengedi, hogy már meglévő rekord helyére vigyünk fel újat, ugyanakkor a még fel nem töltött rekordok is módosíthatók, illetve törölhetők.

Nem érdemes azonban ezeket a funkciókat ilyen egységesen kezelni. Biztonságosabb, ha a funkciókat egymástól jól elhatárolva kezeljük, sőt egy újabbat is célszerű bevezetni. Ez pedig a

— LEKÉRDEZÉS,

melynek során megkeresünk az állományban egy adott rekordot, és beolvassuk az adatait. A lekérdezés lényege a már ismert olvasás: először a rekordra pozicionálunk, majd beolvassuk az adatait.

A karbantartási funkciók a lekérdezéssel kombinálva már lehetővé teszik, hogy csak valóban új rekordot lehessen felvinni, és csak valóban létezőt lehessen módosítani, illetve törölni.

A karbantartást mindig a LEKÉRDEZÉS funkcióval kezdjük. Ennek nem feltétlenül kell magában foglalnia az összes adat beolvasását, elég, ha meggyőződik a rekord állapotáról, vagyis arról, hogy az üres, feltöltött vagy törölt-e. Lekérdezéskor általában mégis be szoktuk olvasni a teljes rekordot, hiszen módosításkor, törléskor amúgy is szükség lehet az adatokra. Másrészt a beolvasott adatok a képernyőn is megjeleníthetők, lehetőséget nyújtva a párbeszédes (interaktív) módosításra. Ezenkívül a lekérdezés önálló funkcióként is igényelhető; például ha a rekordot nem akarjuk módosítani, csupán adatokat szeretnénk kinyerni belőle.

A FELVITEL funkciót csak akkor hajtjuk végre, ha a lekérdezés üres vagy törölt rekordot talált, vagy ha nem találta meg a rekordot. Ilyenkor bekérjük a rekord adatait, összeállítjuk a rekordot, és felírjuk az állományba. Lehetőséget adhatunk a gépkezelőnek arra is, hogy a felvitelbe beavatkozhasson, azaz a felírás előtt az adatokat tetszés szerint javíthassa, illetve hogy végső soron akár az egész felvitelt is letilthassa.

A MÓDOSÍTÁS funkciót csak akkor hajtjuk végre, ha a lekérdezés feltöltött rekordot talált. Ilyenkor kiírjuk a rekordban tárolt adatokat, és lehetővé tesszük, hogy a gépkezelő tetszés szerint, akár többször is módosíthassa őket, majd a gépkezelő megfelelő jelzésére az így módosult rekordot visszaírjuk az állományba.

A TÖRLÉS funkciót csak akkor hajtjuk végre, ha a lekérdezés feltöltött rekordot talált. Ilyenkor kiírjuk a rekordban tárolt adatokat, és a gépkezelőtől a törlés végrehajtását megerősítő választ várunk, nehogy tévedésből olyan rekordot töröljünk, amelyre még szükség van. Megfelelő válasz esetén egyezményes módon felülírjuk a rekordot. Itt alapvetően két lehetőség közül választhatunk: vagy úgy írjuk felül a rekordot, hogy ezzel egy üres rekordot állítunk elő; vagy úgy, hogy a rekord adatai megmaradnak, és csak

egy megállapodás szerinti törlőjel (állapotjelző) utal a rekordban arra, hogy törölt rekordról van szó.
E funkciók persze további kérdéseket vetnek fel. Ha törléskor megőrizzük a rekord adatait, akkor milyen törlőjelet használjunk, és ez hol legyen a rekordban? Új rekordot vihetünk-e fel a töröltek helyére? Ha igen, mi értelme megőrizni a törölt adatokat? Ha nem, akkor az így megmaradó és az állományban hasznos tárhelyet lekötő törölt rekordokat meddig őrizzük meg?
Mindezekre a kérdésekre nem adható általános válasz. Minden esetben a konkrét állomány és a konkrét feldolgozási igények ismeretében, egyedileg kell döntenünk. Ehhez a bemutatott példák, melyek során e kérdésekre még visszatérünk, némi támpontot adnak.
Ugyanúgy, mint a random állományoknál, itt is valamilyen összefüggést kell találnunk vagy létesítenünk a rekord és a tárolási helye között.
A random állományoknál bemutatott leképezési módok, még az indextábla is, alkalmazható a relatív állományokra, azzal az eltéréssel, hogy az ottani blokkcímek helyett a rekordsorszámokat kell szerepeltetnünk. (Nem érdemes azonban a random leképezési módokat másolnunk, mivel a relatív leképezés általában egyszerűbb.)
A bonyolultabb leképezéseknek külön fejezetet szentelünk, ezért itt most egy olyan példát mutatunk be, ahol a leképezés a lehető legegyszerűbb: minden rekord az érkezése sorrendjében kap egy sorszámot, és ez fogja meghatározni a helyét az állományban, teljesen függetlenül a rekord azonosítójától. Ahhoz az elterjedt szokáshoz hasonlóan, hogy a beérkező bizonylatoknak, leveleknek stb. folyamatosan növekvő bizonylatszámot, iktatószámot adunk.)
Legyen tehát egy eszköznyilvántartásunk, amelyben az egyszerűség kedvéért csak a következő adatokat tároljuk:

— az eszköz megnevezése,
— az eszköz gyári száma,
— az eszköz értéke.

A rekorddal kapcsolatban állapodjunk meg abban, hogy az adatok közé a (soros és random állományoknál) már megszokott szeparátorjelet, a RETURN, azaz CHR$(13) karaktert helyezzük el; továbbá a rekordban külön állapotjelzőt használunk a rekord állapotának meghatározására. Ezt az állapotjelet a rekord első bájtján tároljuk, a tartalma pedig egy π, egy plusz- vagy egy mínuszjel, attól függően, hogy a rekord üres, feltöltött vagy törölt.
Ennek megfelelően a rekord ilyen lesz:

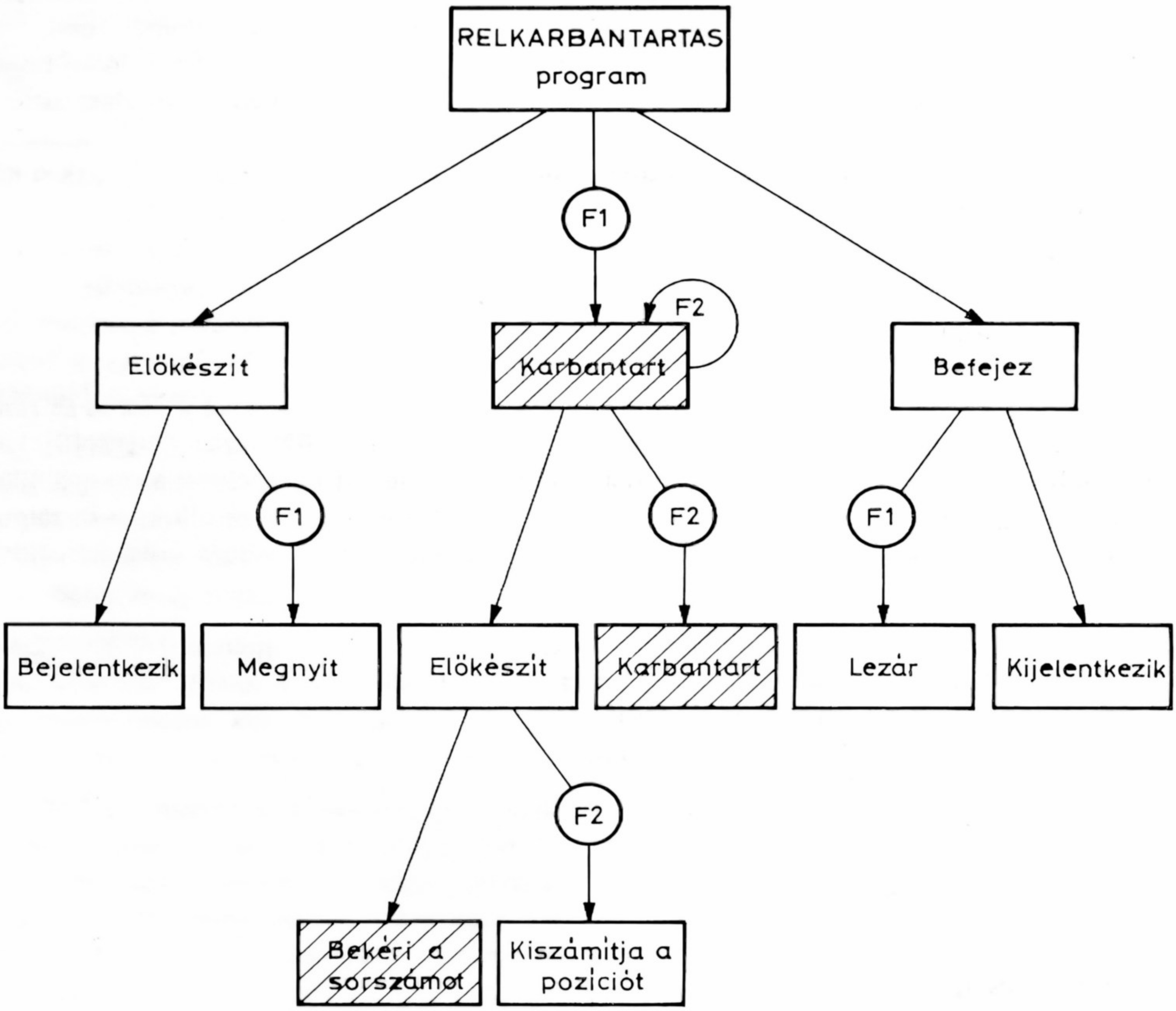
— T$: állapotjelző = 1 bájt (π vagy "+" vagy "—")
— S$: szeparátorjel = 1 bájt (CHR$(13))
— M$: megnevezés = 20 bájt
— S$: szeparátorjel = 1 bájt (CHR$(13))
— G$: gyári szám = 8 bájt
— S$: szeparátorjel = 1 bájt (CHR$(13))
— E$: érték = 6 bájt
— S$: szeparátorjel = 1 bájt (CHR$(13))

A rekordhossz így összesen 39 bájt.

Megjegyezzük, hogy az állapotjelző elvileg akárhol lehetne a rekordban, akár a végén is. Tulajdonképpen nincs is rá feltétlenül szükség. Ugyanis az üres rekordot jelezhetné az első bájton a π, a feltöltöttet pedig az, hogy ott bármi más áll. A törölt rekordot pedig jelezhetné az is, hogy az eszköz megnevezése helyett a "STORNO" szöveg szerepel. Nem célszerű azonban, hogy a rekord adatmezőinek kettős szerepe legyen: egyrészt az adatok tárolása, másrészt a rekord állapotának jelzése. Még zavaróbb, ha az utóbbi nem egy, hanem több adatmező tartalmától is függ. A külön állapotjelző mindössze 2 bájttal növeli meg a rekordméretet, így hacsak nem vagyunk különösen helyszűkében, ne sajnáljuk a funkciók világos szétválasztása és könnyebb kezelhetősége érdekében ezt a rekordra ,,pazarolni".

A rekordkezelés szabályait figyelmen kívül hagyva szeparátorjelként a sorvége jelet használjuk, csak azért, mert így a hibásan létrehozott rekordok is adatonként olvashatók lesznek teszteléskor. A program egyébként helyesen működne S$=CHR$(44) szeparátorjellel, azaz vesszővel is. Célszerű, hogy tesztelés után a jelet ennek megfelelően cseréljük ki. (Mindezek a később bemutatandó programokra is vonatkoznak.)

Most pedig lássuk az állomány karbantartásának szerkezetét. Nyilvánvalóan ez a korábban ismertetett karbantartási funkciókra épül (31. ábra).

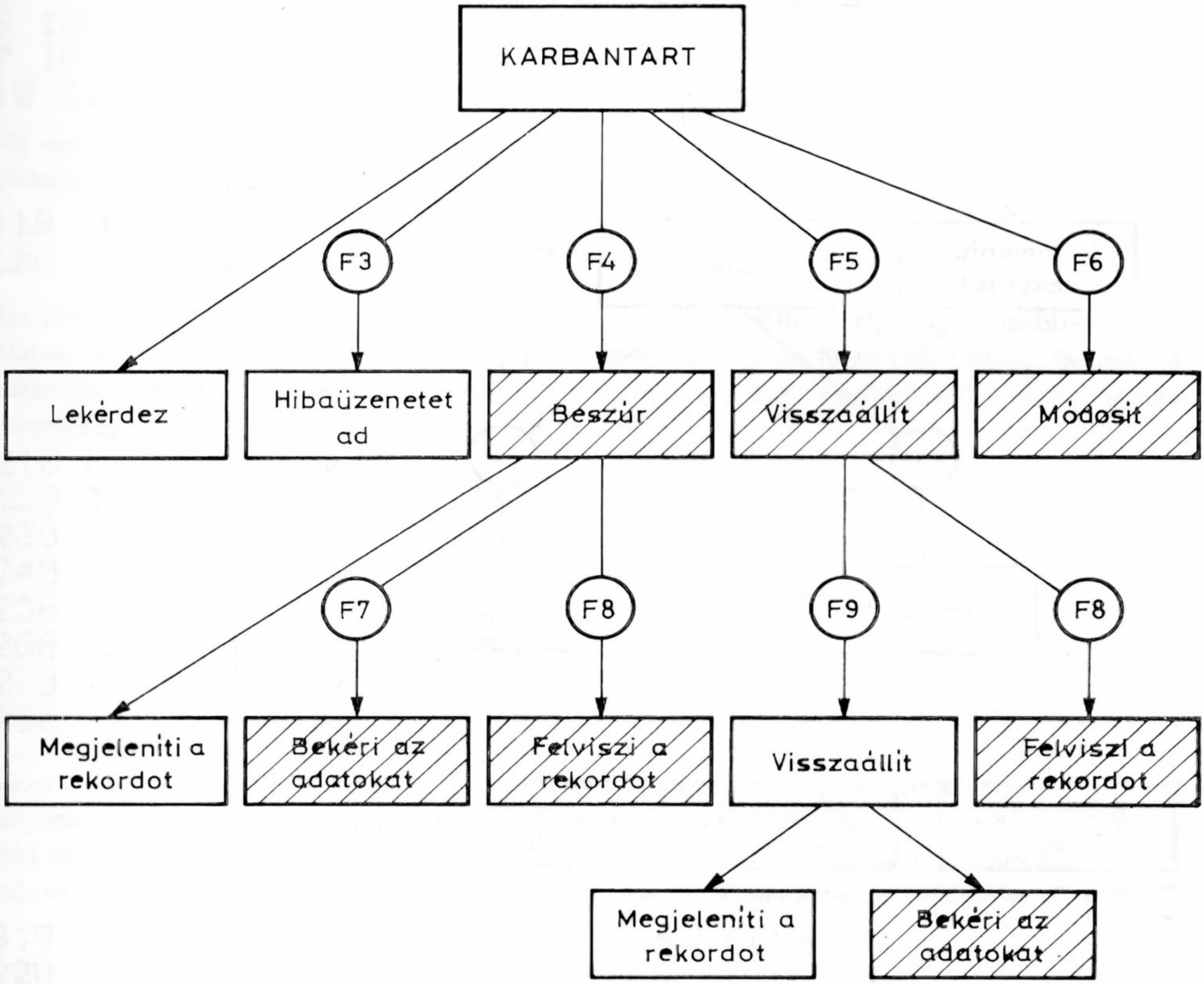


31. ábra. Relatív állomány karbantartása I.

Ahol a feltételek:

- F1: ha a program elindítható, azaz a "MEHET?" kérdésre adott V$ válasz="I";
- F2: ha van karbantartás, azaz a begépelt N sorszám nem 0.

Maga a karbantartás a felvitelt (beszúrást), módosítást, törlést két járulékos funkcióval egészíti ki: a hibás eset kezelésével és a törölt rekord helyreállításával. Itt látható a külön állapotjelző egyik előnye: a törölt rekord adatai megtarthatók, és a jelzőbájt átállításával a rekord helyreállítható (32. ábra).



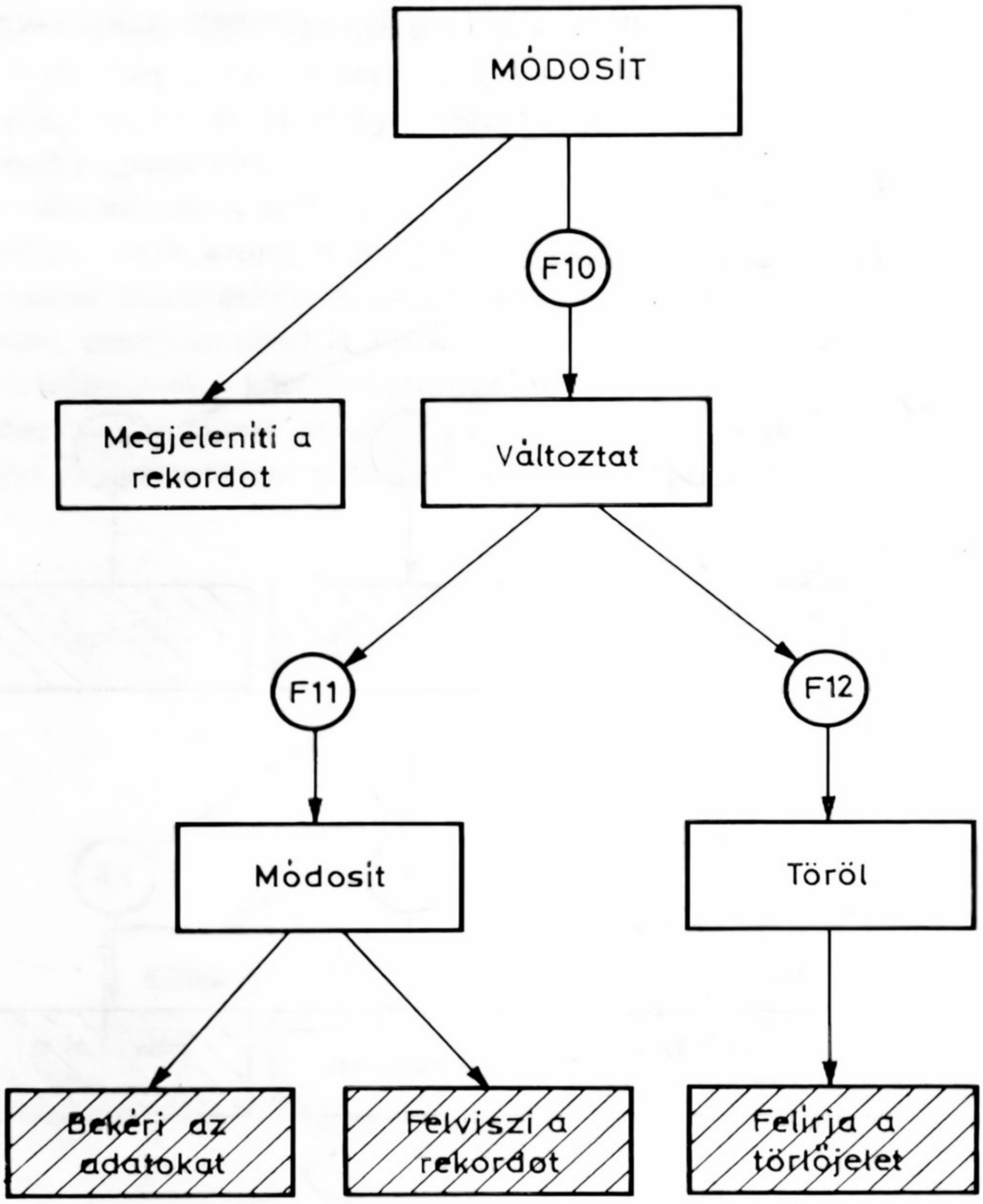
32. ábra. Relatív állomány karbantartása II. – KARBANTART

A feltételek:

- F3: ha a pozicionálás sikertelen volt, azaz a H hibakód > 20, de nem = 50;
- F4: ha a rekord nincs az állományban, tehát a H hibakód = 50;
- F5: ha a rekord törölve volt, vagyis az előző feltételek nem teljesülnek, és a T$ állapotjelző = "–";
- F6: ha a rekord létezik, és nincs törölve, azaz az előző feltételek nem teljesülnek, és a T$ állapotjelző = "+";

– F7: ha a rekord beszúrható, vagyis a V$ válasz = "I";
– F8: ha a rekord felvihető, tehát a V$ válasz = "I";
– F9: ha a törölt rekord visszaállítható, azaz a V$ válasz = "I".

A módosítás három részből áll: lekérdezés (ha nem módosítunk), módosítás, törlés (33. ábra).



33. ábra. Relatív állomány karbantartása III. – MÓDOSÍT

A feltételek:

– F10: ha nem csak lekérdezést akarunk végrehajtani, vagyis a V$ válasz = "N";
– F11: ha a rekord módosítandó, azaz a V$ válasz = "I";
– F12: ha a rekord törlendő, tehát a V$ válasz = "I".

Megjegyezzük, hogy nem is olyan ritka az az igény, hogy a törölt rekordokat meg kell őrizni, és hogy a rekordoknak sorszámuk legyen. Ugyanis meglehetősen sok az olyan dokumentum (bizonylat), amelynek a kezelésére szigorú előírások vannak. Ilyenek például a kimenő számlák. A számlaszámoknak meghatározott számtól kezdve, egyesével kell növekedniük. A rontott számlák nem semmisíthetők meg, hanem megfelelő érvénytele-

nítéssel megőrzendők, és a számuk sem adható ki más számláknak. Ilyesfajta nyilvántartás kezelésére a relatív állomány kiválóan alkalmas.

Következzék tehát a program, amely először is bejelentkezik. Ekkor a karbantartás még leállítható:

```
1 PRINT"❐":PRINT:PRINT
2 PRINT "      ▣                    ■"
3 PRINT "      ▣ ESZKOZNYILVANTARTAS ■"
4 PRINT "      ▣                    ■"
5 PRINT:PRINT:PRINT
6 INPUT " ▣*■ MEHET =I/N=  *■■■■";V$
7 IF V$<>"I" THEN GOTO 910
10 S$=CHR$(13)
```

Ha nem állítottuk le, a program megnyitja az "ESZKOZOK" állományt, és persze a pozicionáláshoz szükséges parancscsatornát is:

```
110 OPEN 15,8,15
120 OPEN  2,8, 2,"ESZKOZOK"
```

Ezután bekéri a karbantartandó rekord N sorszámát. Ellenőrzi, hogy kisebb-e 0-nál, illetve nagyobb-e 999-nél. Ha igen, új sorszámot kér. Ha az N sorszám nulla, befejezi a karbantartást. Ha nem, a pozicionáláshoz szükséges kétszer egybájtos RH és RL alakra konvertálja:

```
210 PRINT"❐":PRINT
220 INPUT " ▣*■ SORSZAM =    0■■■■";N$
230 :  N=INT(VAL(N$))
240 :  :  IF N<0   THEN GOTO 210
250 :  :  IF N>999 THEN GOTO 210
260 :  :  IF N=0   THEN GOTO 810
270 :  RH=INT(N/256)
280 :  RL=N-RH*256
```

Most következik a lekérdezés. A program az N sorszámú rekordra pozicionál, és a P = 1 bájtpozíciójáról beolvassa a T$ állapotjelzőt. Közben hibavizsgálatot is végez. A H hibakód és a T$ állapotjelző tartalmától függően végrehajtja a különböző karbantartási funkciókat:

```
310 :  P=1: GOSUB 1010 :REM==> OLVASAS
320 :  IF H>20 AND H<>50 THEN GOTO 410
330 :  IF H=50           THEN GOTO 510
340 :  IF T$="π"         THEN GOTO 510
350 :  IF T$="-"         THEN GOTO 610
360 :  IF T$="+"         THEN GOTO 710
399 GOTO 210
```

Ha a rekord pozicionálása nem sikerült, hibaüzenetet ad, amely tartalmazza a rekord N$ sorszámát, a gépi H hibakódot és H$ hibaüzenetet, valamint a T sávcímet és a B szektorcímet. Ezek után dönthetünk, hogy a további karbantartásnak van-e értelme. Ha igen, a program új rekordsorszámot kér. Egyébként leáll:

```
410 :  PRINT:PRINT
420 :  PRINT " ▣ *** HIBA *** ■"
430 :  PRINT:PRINT " REKORD = ";N$
440 :  PRINT:PRINT H;H$,T;B
450 :  PRINT:PRINT " ▣*■ MEHET = I/N=  N";
460 :  INPUT "▮▮▮▮";V$
470 :  IF V$<>"I" THEN GOTO 810
499 GOTO 210
```

Ha a rekord nem szerepel az állományban, vagy szerepel, de üres, a program feltételezi, hogy beszúrás (új felvitel) következik. Ennek megfelelően megjeleníti a rekordtartalmat a képernyőn. Ilyenkor persze minden adat üres. Ha úgy döntünk, hogy a rekord beszúrható, akkor a program kéri az adatokat. Ezeket a megfelelő rovatok kitöltésével adjuk meg. Az adatbegépelés után ellenőrzési lehetőségünk van. Ha úgy döntünk, hogy a rekord az adott adatokkal felvihető, a program felírja a rekordot az állományba. Ezután, illetve minden más esetben új rekordsorszámot kér:

```
510 :  GOSUB 2010 :REM==> MEGJELENITES
520 :  PRINT:PRINT
521 :  PRINT " ▣*■ BESZURHATO =I/N=  N";
530 :    INPUT "▮▮▮▮";V$
540 :  IF V$<>"I" THEN GOTO 599
550 :  :  GOSUB 3010 :REM==> ADATBEKERES
560 :  :  PRINT:PRINT
561 :  :  PRINT " ▣*■ FELVIHETO  =I/N=  N";
570 :  :  INPUT "▮▮▮▮";V$
580 :  IF V$<>"I" THEN GOTO 599
590 :  :  T$="+": GOSUB 4010 :REM==> IRAS
599 GOTO 210
```

Ha a lekérdezés törölt rekordot talált, a program ilyen értelmű jelzést ad, majd megkérdezi, hogy a rekord visszaállítható-e. Ha tagadó választ adunk, új rekordsorszámot kér. Egyébként beolvassa a rekord P = 3 bájtpozíciójától kezdve az adatokat, megjeleníti őket a képernyőn, majd lehetőséget ad a módosításukra is. Ezután dönthetünk, hogy a rekord ebben a formában felvihető-e. Ha igen, a program felírja a rekordot az állományba, persze most már a "+" jelre átállított állapotjelzővel. Egyébként meghagyja a rekordot az eredeti (törölt) állapotában. A funkció végrehajtása után új rekordsorszámot kér:

```
610 :  PRINT:PRINT
620 :  PRINT " ▣ *** TOROLVE *** ■"
630 :  PRINT:PRINT
631 :  PRINT " ▣*■ VISSZAALLITHATO";
632 :  INPUT " =I/N=  N▮▮▮▮";V$
633 :  IF V$<>"I" THEN GOTO 699
640 :  : P=3: GOSUB 1010 :REM==> OLVASAS
650 :  :  GOSUB 2010 :REM==> MEGJELENITES
660 :  :  GOSUB 3010 :REM==> ADATBEKERES
```

```
670 :  :  PRINT:PRINT
671 :  :  PRINT " ■*■ FELVIHETO =I/N=  N";
672 :  :  INPUT "■■■■";V$
673 :  IF V$<>"I" THEN GOTO 699
680 :  :  T$="+": GOSUB 4010 :REM==> IRAS
699 GOTO 210
```

Ha viszont a rekord létezik az állományban, és fel is van töltve, de nem törölt, akkor a program rátér a módosításra. Ez azzal kezdődik, hogy beolvassa a P = 3 bájtpozíciótól kezdve a rekord adatait, majd megjeleníti őket a képernyőn. Eddig voltaképpen egy lekérdezést hajtottunk végre. Ha csak ez volt a célunk, és kérjük a következő rekordot, a program áttér az új rekordsorszám bekérésére:

```
710 :  P=3: GOSUB 1010 :REM==> OLVASAS
720 :  GOSUB 2010 :REM==> MEGJELENITES
730 :  PRINT:PRINT
731 :  PRINT " ■*■ JOHET A KOVETKEZO";
732 :  INPUT " =I/N=  I■■■■";V$
733 :  IF V$<>"N" THEN GOTO 799
```

Egyébként el kell döntenünk, hogy a rekordot módosítani vagy törölni akarjuk-e. Ha az előbbit választjuk, a program bekéri a módosító adatokat, majd ezek begépelése után felviszi a módosított rekordot az állományba:

```
740 :  PRINT "■";
741 :  PRINT " ■*■ MODOSITANDO        ";
742 :  INPUT " =I/N=  N■■■■";V$
743 :  IF V$<>"I" THEN GOTO 780
750 :  :  GOSUB 3010 :REM==> ADATBEKERES
755 :  :  PRINT:PRINT:PRINT
760 :  :  T$="+": GOSUB 4010 :REM==> IRAS
770 :  GOTO 799
```

Ha viszont a törlést választottuk, akkor a T$ állapotjelzőt átállítja a "–" törlőjelre, majd így írja vissza a teljes rekordot az állományba. Ezáltal a törölt rekord adatai nem semmisülnek meg:

```
780 :  PRINT "■";
781 :  PRINT " ■*■ TORLENDO           ";
782 :  INPUT " =I/N=  N■■■■";V$
783 :  IF V$<>"I" THEN GOTO 799
790 :  :  T$="-": GOSUB 4010 :REM==> IRAS
799 GOTO 210
```

A program minden egyes karbantartási funkció végrehajtása után az új karbantartandó rekordsorszám bekérésére tér vissza. Ha viszont ennek értéke nulla, befejezi a karbantartást, azaz lezárja az állományt és a parancscsatornát:

```
810 CLOSE  2
820 CLOSE 15
```

Majd elbúcsúzik és leáll. Ugyanezt teszi, ha a karbantartás elindítását letiltottuk:

```
910 PRINT"⊐":PRINT:PRINT
920 PRINT "     ▣       ■"
930 PRINT "     ▣ VEGE ■"
940 PRINT "     ▣       ■"
950 PRINT:PRINT:PRINT
999 END
```

Az önálló funkciójú tevékenységeket külön szubrutinok hajtják végre.

Közülük első az olvasás. Ez a T$ állapotjelző, az M$ megnevezés, a G$ gyári szám és az E$ érték törlésével kezdődik. Majd a pozicionálás következik az N rekordsorszám RL és RH alakban megadott értékének megfelelő rekordra, itt is a P bájtpozícióra. A parancscsatorna lekérdezésével a szubrutin ellenőrzi a pozicionálás sikerességét. Hiba esetén az olvasást nem hajtja végre. Sikeres pozicionálás után a szubrutin vagy csak a T$ állapotjelzőt, vagy a rekord többi, M$, G$, E$ adatát olvassa be, a paraméterként megadott P bájtpozíció értékétől függően:

```
1010 T$="π"
1020 M$=" "
1030 G$=" "
1040 E$=" "
1110 PRINT#15,"P"CHR$(2)CHR$(RL)CHR$(RH)CHR$(P)
1120 INPUT#15,H,H$,T,B
1130 IF H>19 THEN GOTO 1999
1210 :   IF P=1 THEN INPUT#2,T$
1310 :   IF P=3 THEN INPUT#2,M$,G$,E$
1999 RETURN
```

Olvasáskor már nem kérdezzük le a parancscsatornát. Ha a pozicionálás sikerült, az olvasás sikertelensége valószínűtlen, hiszen az állapotjelzőt is figyeljük. (Lásd RELOLVASAS!)

A következő szubrutin a rekordot jeleníti meg a képernyőn. Ez abból áll, hogy fejlécként kiírja a rekord N sorszámát, majd az eszköz M$ megnevezését, G$ gyári számát és E$ értékét:

```
2010 PRINT"⊐":PRINT
2020 PRINT:PRINT " SORSZ. =   ";N
2030 PRINT " ---------------";
2031 PRINT "----------------"
2040 PRINT:PRINT " MEGNEV =   ";
2041 PRINT "▣                    ■";"◆"
2042 PRINT "⊐";TAB(12);"▣";M$;"■"
2050 PRINT:PRINT " GYSZAM =   ";
2051 PRINT "▣          ■";"◆"
2052 PRINT "⊐";TAB(12);"▣";G$;"■"
2060 PRINT:PRINT " ERTEKE =   ";
2061 PRINT "▣      ■";"◆"
2062 PRINT "⊐";TAB(12);"▣";E$;"■"
2999 RETURN
```

A rekord adatait a formanyomtatványoknál szokásos kiemelt rovatokban adja meg. Minden rovat meghatározza az adat maximális méretét, amire a rovat végén álló rombusz (kárójel) még külön is figyelmeztet. Ezek a rovatok különösen hasznosak az adatok bekérésekor, amire új rekord felvitele vagy meglévő rekord módosítása esetén kerül sor. Ez a szubrutin ugyanis a billentyűzetről bekéri az M$ megnevezést, a G$ gyári számot és az E$ értéket:

```
3010 PRINT "■■■■■■■■■";TAB(10);
3020 INPUT M$
3030 M$=LEFT$(M$,20)
3040 PRINT "■";TAB(10);
3050 INPUT G$
3060 G$=LEFT$(G$,8)
3070 PRINT "■";TAB(10);
3080 INPUT E$
3090 E$=LEFT$(E$,6)
3999 RETURN
```

Erre azonban mindig csak a rekord megjelenítése után kerülhet sor. Így az adatbekérés során a képernyőn megjelenő rovatokat kell kitöltenünk, illetve felülírnunk, ami az adatbevitelt és a módosítást rendkívül megkönnyíti.

Végül külön szubrutinnal írjuk fel a rekordot az állományba:

```
4010 IF H=50 THEN GOTO 4110
4020 :  PRINT#15,"P"CHR$(2)CHR$(RL)CHR$(RH)CHR$(1)
4030 :  INPUT#15,H,H$,T,B
4040 :  IF H>19 THEN GOTO 4210
4110 PRINT#2,T$;S$;M$;S$;G$;S$;E$
4120 IF T$="+" THEN U$="FELVIVE"
4130 IF T$="-" THEN U$="TOROLVE"
4140 PRINT:PRINT
4150 PRINT " ■ *** ";U$;" *** ■"
4199 GOTO 4310
4210 :  PRINT:PRINT
4220 :  PRINT " ■ *** HIBA *** ■"
4230 :  PRINT
4240 :  PRINT H;H$,T;B
4310 T1=TIME
4320 T2=TIME
4330 IF T2-T1<120 THEN GOTO 4320
4999 RETURN
```

Mindenekelőtt meg kell néznünk, hogy meglévő rekordba kell-e az adatokat felírnunk, vagy az állomány kiterjesztésével új rekordot kell létrehoznunk. Ez a lekérdezéskor beállított H hibakód alapján, a 4010-es sorban egyértelműen eldönthető.

A kérdés azért nagyon fontos, mert a sikeres olvasás után, tehát létező rekord esetén az íráshoz újra kell pozicionálnunk. Mivel mindig teljes rekordot írunk fel, ez a 4020—4040

sorokban mindig az első bájtra történik. Sikertelen pozicionálás esetén a szubrutin meg sem kíséreli az írást, viszont a 4210–4240 sorokban hibaüzenetet ad. Sikeres pozicionálás esetén a 4110-es sor felírja a rekordot az állományba. (Az olvasáshoz hasonlóan az írás sikerességét ilyenkor már nem vizsgálja.) Közvetlenül e művelet végrehajtása történik meg akkor, ha a rekord nem szerepel az állományban, vagyis amikor bővíteni kell. Ugyanis ilyenkor a lekérdezésnél végrehajtott pozicionálást nem követte olvasás, így az íráshoz nem kell újra pozicionálni, sőt nem is szabad, mert kétszer egymás után pozicionálni tilos.

FIGYELEM!

Ha egy rekordra kétszer egymás után pozicionálunk, még ha különböző bájtokra is, a második pozicionáláskor 1-es hibakóddal **NO CHANNEL** hibaüzenetet kapunk. Ezt alkalomadtán kipróbálhatjuk, ha a RELOLVASAS programban az 510-es pozicionáló utasítást az 515-ös sorban megismételjük, majd a programot a "PROBA" állományra lefuttatjuk.

Az írás végrehajtása után a 4120–4199 sorokban a szubrutin megfelelő üzenetet ad. Hogy legyen idő ezt elolvasni, mielőtt továbblépne, mintegy 2 másodpercet vár. A várakozási időt a 4310–4999 sorokban a gépi óra lekérdezésével méri.

Ezzel a program kész. Ha begépeltük, mentsük ki RELKARBANTARTAS néven, de ne indítsuk el, mert az "ESZKOZOK" állomány még nincs létrehozva, márpedig karbantartani csak létező állományt lehet.

Mielőtt kipróbálnánk a programot, fordítsuk még figyelmünket a törölt rekord helyreállítására. Ezzel kapcsolatban két lehetőségünk van: a rekordot vagy változatlan, vagy módosított formában állítjuk helyre. Programunkban az utóbbi megoldást választottuk. A jobb áttekinthetőség kedvéért egybefüggően is közöljük a vezérlőprogram listáját (tehát a szubrutinok nélkül):

```
1 PRINT"□":PRINT:PRINT
2 PRINT "     ■                       ■"
3 PRINT "     ■ ESZKOZNYILVANTARTAS ■"
4 PRINT "     ■                       ■"
5 PRINT:PRINT:PRINT
6 INPUT " ■■■ MEHET =I/N=  ■■■■■";V$
7 IF V$<>"I" THEN GOTO 910
10 S$=CHR$(13)
100 REM ---------------------------------MEGNYITAS
110 OPEN 15,8,15
120 OPEN  2,8, 2,"ESZKOZOK"
200 REM ---------------------------SORSZAMBEKERES
210 PRINT"□":PRINT
220 INPUT " ■■■ SORSZAM =    ■■■■■";N$
230 :   N=INT(VAL(N$))
240 :   :   IF N<0   THEN GOTO 210
250 :   :   IF N>999 THEN GOTO 210
260 :   :   IF N=0   THEN GOTO 810
270 :   RH=INT(N/256)
280 :   RL=N-RH*256
```

```
300 REM ------------------------------------KARBANTARTAS
310 :   P=1: GOSUB 1010 :REM==> OLVASAS
320 :   IF H>20 AND H<>50 THEN GOTO 410
330 :   IF H=50            THEN GOTO 510
340 :   IF T$="π"          THEN GOTO 510
350 :   IF T$="-"          THEN GOTO 610
360 :   IF T$="+"          THEN GOTO 710
399 GOTO 210
400 REM --------------------------------------SIKERTELEN
410 :   PRINT:PRINT
420 :   PRINT " ◪ *** HIBA *** ■"
430 :   PRINT:PRINT " REKORD = ";N$
440 :   PRINT:PRINT H;H$,T;B
450 :   PRINT:PRINT "  ◪*■ MEHET = I/N=  N";
460 :   INPUT "████";V$
470 :   IF V$<>"I" THEN GOTO 810
499 GOTO 210
500 REM ----------------------------------------BESZURAS
510 :   GOSUB 2010 :REM==> MEGJELENITES
520 :   PRINT:PRINT
521 :   PRINT " ◪*■ BESZURHATO =I/N=  N";
530 : :   INPUT "████";V$
540 :   IF V$<>"I" THEN GOTO 599
550 : :   GOSUB 3010 :REM==> ADATBEKERES
560 : :   PRINT:PRINT
561 : :   PRINT " ◪*■ FELVIHETO  =I/N=  N";
570 : :   INPUT "████";V$
580 :   IF V$<>"I" THEN GOTO 599
590 : :   T$="+": GOSUB 4010 :REM==> IRAS
599 GOTO 210
600 REM ----------------------------------VISSZAALLITAS
610 :   PRINT:PRINT
620 :   PRINT " ◪ *** TOROLVE *** ■"
630 :   PRINT:PRINT
631 :   PRINT " ◪*■ VISSZAALLITHATO";
632 :   INPUT " =I/N=  N████";V$
633 :   IF V$<>"I" THEN GOTO 699
640 : :   P=3: GOSUB 1010 :REM==> OLVASAS
650 : :   GOSUB 2010 :REM==> MEGJELENITES
660 : :   GOSUB 3010 :REM==> ADATBEKERES
670 : :   PRINT:PRINT
671 : :   PRINT " ◪*■ FELVIHETO =I/N=  N";
672 : :   INPUT "████";V$
673 :   IF V$<>"I" THEN GOTO 699
680 : :   T$="+": GOSUB 4010 :REM==> IRAS
699 GOTO 210
```

```
700 REM ------------------------------LEKERDEZES
710 :   P=3: GOSUB 1010 :REM==> OLVASAS
720 :   GOSUB 2010 :REM==> MEGJELENITES
730 :   PRINT:PRINT
731 :   PRINT " ▚*■ JOHET A KOVETKEZO";
732 :   INPUT " =I/N=  I■■■";V$
733 :   IF V$<>"N" THEN GOTO 799
739 REM --------------------------------MODOSITAS
740 :   PRINT "□";
741 :   PRINT " ▚*■ MODOSITANDO         ";
742 :   INPUT " =I/N=  N■■■";V$
743 :   IF V$<>"I" THEN GOTO 780
750 :  :   GOSUB 3010 :REM==> ADATBEKERES
755 :  :   PRINT:PRINT:PRINT
760 :  :   T$="+": GOSUB 4010 :REM==> IRAS
770 :   GOTO 799
779 REM -----------------------------------TORLES
780 :   PRINT "□";
781 :   PRINT " ▚*■ TORLENDO            ";
782 :   INPUT " =I/N=  N■■■";V$
783 :   IF V$<>"I" THEN GOTO 799
790 :  :   T$="-": GOSUB 4010 :REM==> IRAS
799 GOTO 210
800 REM ---------------------------------LEZARAS
810 CLOSE  2
820 CLOSE 15
900 REM ----------------------------KIJELENTKEZES
910 PRINT"□":PRINT:PRINT
920 PRINT "     ▚      ■"
930 PRINT "     ▚ VEGE ■"
940 PRINT "     ▚      ■"
950 PRINT:PRINT:PRINT
999 END
```

Látható, hogy módosítást két helyen is végrehajthatunk. Ugyanis nemcsak a létező feltöltött rekordok módosíthatók a 740–770 sorokban, hanem a 640–670 sorokban a visszaállítással egyidejűleg a törölt rekordok is.

Ha úgy érezzük, hogy ez a felhasználó számára kétségtelenül kényelmes megoldás, de a program áttekinthetősége és biztonságossága szempontjából hátrányos, akkor átírhatjuk a törölt rekordot helyreállító modult úgy, hogy az valóban csak visszaállítani tudjon:

```
610 :   PRINT:PRINT
620 :   PRINT " ▚ *** TOROLVE *** ■"
630 :   PRINT:PRINT
631 :   REM -----TOROLVE-----
632 :   REM -----TOROLVE-----
633 :   REM -----TOROLVE-----
```

```
640 :  P=3: GOSUB 1010 :REM==> OLVASAS
650 :  GOSUB 2010 :REM==> MEGJELENITES
660 :  REM -----TOROLVE-----
670 :  PRINT:PRINT
671 :  PRINT " ■*■ VISSZAALLITHATO =I/N=  N";
672 :  INPUT "■■■■";V$
673 :  IF V$<>"I" THEN GOTO 699
680 :  :  T$="+": GOSUB 4010 :REM==> IRAS
699 GOTO 210
```

Ilyenkor, ha a gépkezelő módosítani is akar, akkor ezt külön menetben kell megtennie: először visszaállítja a rekordot, majd ugyanazt a rekordsorszámot megadva, végrehajtja a módosítást. Ha ritkán van rá szükség, akkor ez valójában nem jelent nagy kényelmetlenséget a felhasználóknak.

Ha akarjuk, írjuk át a programunkat ennek megfelelően. Csupán a 631, 632, 633, 660 sorokat kell törölnünk, és a 671-es sor kérdésének szövegét kell megváltoztatnunk. Akár átírtuk a programot, akár nem, most már kipróbálhatjuk. Először persze hozzuk létre az állományt. Ehhez a RELGENERALAS programra van szükségünk:

```
RELATIV ALLOMANY LETREHOZASA

■ ALLOMANY  NEVE =? ESZKOZOK
■ REKORD  HOSSZA =? 39
■ REKORDOK SZAMA =? 20
■ VEGJEL         =? π

LETREHOZVA
```

Ügyeljünk arra, hogy az adatokat hibátlanul adjuk meg; az állomány neve "ESZKOZOK", a rekordméret 39 bájt, a végjel pedig π legyen, különben a karbantartó programot nem tudjuk majd használni.

A rekordok száma elvileg legfeljebb 999 lehet; ennyit enged meg a karbantartó program. Ezt persze nem kell kihasználnunk. A teszteléshez elég, ha 20 rekordot adunk meg. (Ha akarjuk, ezzel összhangban módosíthatjuk a karbantartó program rekordsorszám-ellenőrző rutinját.)

Ha a RELGENERALAS sikeresen lefutott, akkor betölthetjük, és elindíthatjuk a RELKARBANTARTAS programot.

```
SORSZ. =    1
------------------------------------------
MEGNEV =
GYSZAM =
ERTEKE =

■ BESZURHATO =I/N=? I
```

Először az 1-es sorszámú eszköz adatait adjuk meg. Minthogy az állomány üres, a program beszúrásként fogja kezelni a rekordot.
Adjunk meg tetszőleges adatokat. Például:

```
SORSZ. =   1
------------------------------------------
MEGNEV = ? C64 KOZP.EGYS.███████◆
GYSZAM = ? 44-1111█◆
ERTEKE = ? 20000█◆

█ FELVIHETO  =I/N=? I

█ZZZ█=█WW█=ZZZ█
```

Ugyanígy vigyünk fel még néhány adatot. Például:

```
SORSZ. =   2
------------------------------------------
MEGNEV = ? C64 TAPEGYSEG████████◆
GYSZAM = ? 44-2222█◆
ERTEKE = ? 5000███◆

█ FELVIHETO  =I/N=? I

█ZZZ█=█WW█=ZZZ█

SORSZ. =   3
------------------------------------------
MEGNEV = ? 1541 LEMEZEGYSEG█████◆
GYSZAM = ? 44-3333█◆
ERTEKE = ? 20000█◆

█ FELVIHETO  =I/N=? I

█ZZZ█=█WW█=ZZZ█

SORSZ. =   4
------------------------------------------
MEGNEV = ? MPS-801 NYOMTATO█████◆
GYSZAM = ? 44-4444█◆
ERTEKE = ? 40000█◆

█ FELVIHETO  =I/N=? I

█ZZZ█=█WW█=ZZZ█
```

Eddig csak új felvitelt, azaz beszúrást hajtottunk végre. Ha kívánjuk, vihetünk még fel rekordokat úgy, hogy kihagyunk sorszámokat, vagy akár úgy is, hogy a sorszám túllépje a 20-at, de a karbantartás kipróbálására már ez a négy rekord is elég.

Módosítsuk a 3-as rekordot:

```
SORSZ. =    3
--------------------------------------------------------
MEGNEV =    [illegible]
GYSZAM =    [illegible]
ERTEKE =    [illegible]

 JOHET A KOVETKEZO =I/N=? N
```

Írjuk át az értéket 20000-ről 30000-re:

```
SORSZ. =    3
--------------------------------------------------------
MEGNEV = ? [illegible]
GYSZAM = ? [illegible]
ERTEKE = ? 3[illegible]

 MODOSITANDO          =I/N=? I

[illegible]
```

Azokat a rovatokat, amelyeket nem módosítunk, hagyjuk változatlanul, vagyis csak a RETURN gombot nyomjuk meg.

Töröljük a 2-es rekordot:

```
SORSZ. =    2
--------------------------------------------------------
MEGNEV =    [illegible]
GYSZAM =    [illegible]
ERTEKE =    [illegible]

 TORLENDO             =I/N=? I

[illegible]
```

Állítsuk vissza a 2-es rekordot:

```
 SORSZAM = ? 2

[illegible]

 VISSZAALLITHATO =I/N=? I
```

```
SORSZ. =    2
----------------------------------------------------------------
MEGNEV = ?
GYSZAM = ?
ERTEKE = ?

FELVIHETO =I/N=? I
```

Kérdezzük le a 4-es rekordot:

```
SORSZ. =    4
----------------------------------------------------------------
MEGNEV =
GYSZAM =
ERTEKE =

JOHET A KOVETKEZO =I/N=? I
```

Örömmel nyugtázhatjuk, hogy a program úgy működik, ahogyan terveztük.

VÁLOGATÁS A RELATÍV ÁLLOMÁNYBAN

A relatív állományban igen kényelmesen lehet válogatni a rekordsorszámok mentén végrehajtott sorozatos közvetlen elérésekkel.

Azonban általában csak soros keresés hajtható végre, ugyanúgy, mint a soros rendezetlen állományoknál (34. ábra).

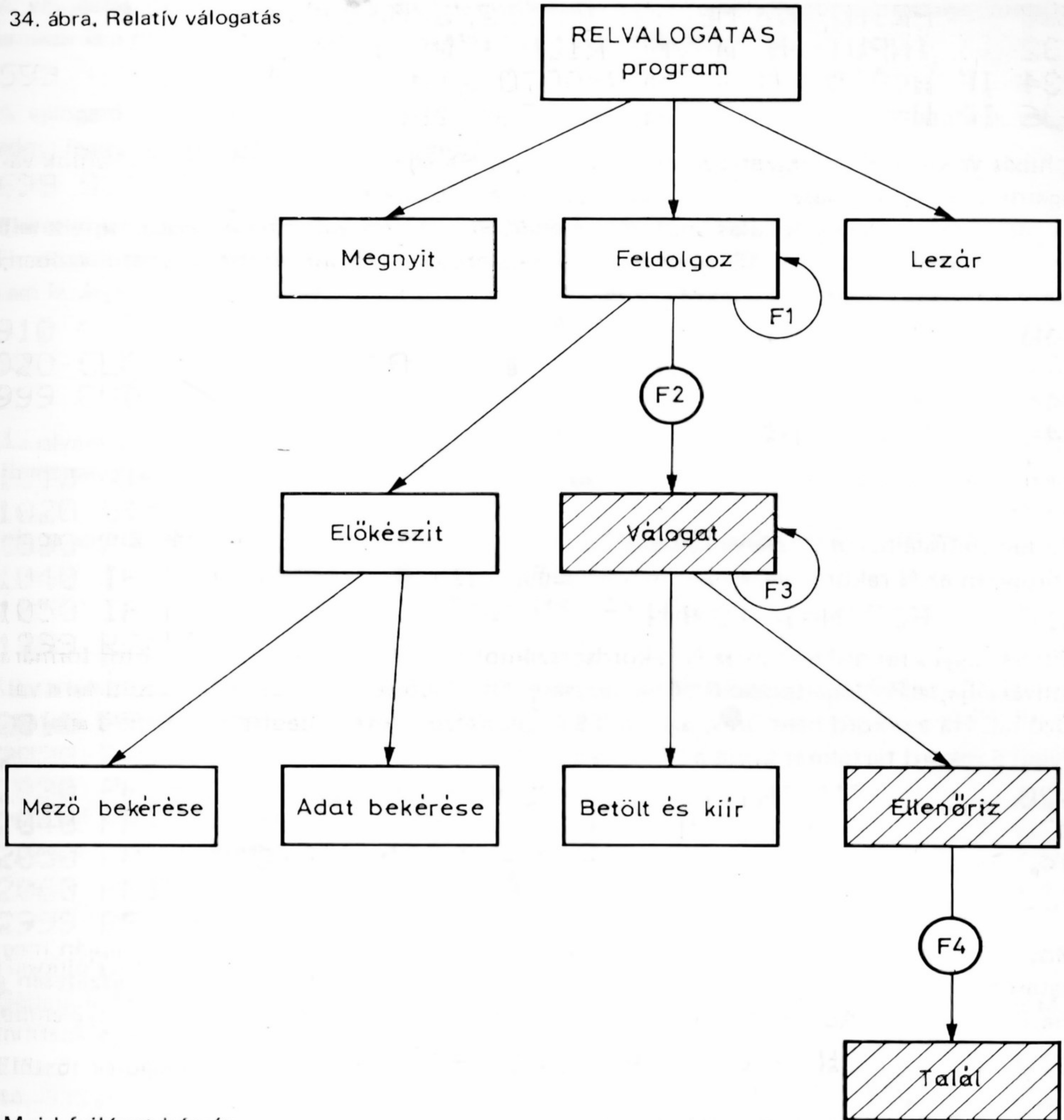
A feltételek:

- F1: ha van kiválasztott mező, azaz W nem = 0;
- F2: ha van kiválasztott mező és válogatandó adat is, tehát W nem = 0 és A$ nem = "–";
- F3: végrehajtandó 1-től NN-ig, egyesével;
- F4: ha a válogatandó adat megegyezik a kiválasztott mező tartalmával, vagyis A$=X$.

Mivel a válogatás alapelveit a „SOROS LEMEZÁLLOMÁNYOK" című kötetben már tárgyaltuk, a program szerkezetéhez nem kell magyarázat. Maga a program hasonlóan egyszerű.
Azzal kezdődik, hogy megnyitja az "ESZKOZOK" állományt, a parancscsatornát, és beállítja a kezdőértékeket:

```
110 PRINT"♥"
120 OPEN 15,8,15:OPEN 2,8,2,"ESZKOZOK"
130 K$="----------------------------------------------------------------"
140 NN=20
```

34. ábra. Relatív válogatás



Majd fejlécet készít:

```
210 PRINT"□"
212 :   PRINT"VALOGATAS AZ ESZKOZOK ALLOMANYBAN"
214 :   PRINT K$
```

Ezután bekéri, hogy melyik mező szerint válogasson; az M$ megnevezés, a G$ gyári szám vagy az E$ érték szerint:

```
220 :   PRINT"█<1> MEGNEVEZES SZERINT"
222 :   PRINT"█<2> GYARI SZAM SZERINT"
224 :   PRINT"█<3> ERTEK      SZERINT"
226 :   PRINT"█<0> --- EGYIK  SEM ---"
228 :   PRINT K$
```

```
230 :   PRINT"▣? MELYIK =/1/2/3/0/= ";
232 :   INPUT W$:W=VAL(RIGHT$(W$,1))
234 IF W<0 OR W>3 THEN GOTO 210
236 IF W=0        THEN GOTO 910
```

A hibás W választ természetesen nem fogadja el. Ha egyik mező szerint sem akarunk válogatni, azaz W = 0 választ adunk, akkor befejezi a válogatást.

Ha meghatároztuk a válogatás során figyelendő mezőt, megadhatjuk a válogatási feltételt is, azaz megadhatunk egy A$ adatot, amely ha szerepel az imént meghatározott mezőben, akkor a rekord kielégíti a válogatási feltételt:

```
240 :   PRINT K$
242 :   INPUT"▣? ADAT =    -████";A$
244 IF A$="-"      THEN GOTO 210
246 :   L=LEN(A$):PRINT"□"
```

Ha nem adunk meg válogatási feltételt, azaz ha az A$ = "–", akkor nem válogat, hanem új mezőt kér.

Miután definiáltuk a W és A$ válogatási szempontot, elkezdődik a válogatás. Ennek során a program az N rekordsorszámok mentén végigmegy a teljes állományon:

```
310 :   FOR N=1 TO NN
```

Minden egyes rekord esetén az N rekordsorszámot az RH és RL kétszer egybájtos formára konvertálja, a P = 1 bájtpozícióról beolvassa a T$ állapotjelzőt, és szóközzel tölti fel a változókat. Ha a rekord nem üres, azaz a T$ állapotjelző nem = π, beolvassa a rekord adatait.

Végül a rekord tartalmát kiírja a képernyőre:

```
320 :   :   RH=INT(N/256):RL=N-RH*256
330 :   :   P=1:GOSUB 1010
340 :   :   IF T$<>"π" THEN P=3:GOSUB 1010
350 :   :   GOSUB 2010
```

Most következik a válogatási szempont ellenőrzése. A program a W válasz alapján meghatározott mező X$ tartalmát összehasonlítja a keresendő A$ adattal. Természetesen a mezőből csak az A$ adat L hosszának megfelelő számú első L bájtot veszi figyelembe:

```
410 :   :   ON W GOTO 430,440,450
420 :   GOTO 599
430 :   :   X$=M$:GOTO 460
440 :   :   X$=G$:GOTO 460
450 :   :   X$=E$:GOTO 460
460 :   :   X$=LEFT$(X$,L)
470 :   IF X$<>A$ THEN GOTO 599
```

Ha az X$ és az A$ adat nem egyezik meg, a rekord nem elégíti ki a válogatási szempontot, ilyenkor a program áttér a következő rekordra. Ellenkező esetben a válogatást megállítja, hogy a rekord tartalmát a képernyőn tetszés szerinti ideig tanulmányozhassuk:

```
510 :   :   PRINT"▣";K$
520 :   :   PRINT"▣TOVABBLEPES BARMELY GOMBBAL!"
530 :   :   GET B$:IF B$="" THEN GOTO 530
```

A válogatás csak akkor folytatódik, ha a billentyűzet valamelyik gombját megnyomjuk, amikor is a program rátér a soron következő rekord feldolgozására:

```
599 : NEXT N
```

A válogatás mindaddig tart, amíg el nem éri az állomány végét, vagyis az általunk megadott legmagasabb NN rekordsorszámot:

```
699 GOTO 210
```

Ekkor a program új válogatási szempontot, azaz W mezőt és A$ adatot kér.
Ha nem kívánunk tovább válogatni az állományból, azaz W = 0 mezőt adunk meg, a program lezárja az "ESZKOZOK" állományt és a parancscsatornát, majd leáll:

```
910 PRINT"□"
920 CLOSE 2:CLOSE 15
999 END
```

Az olvasást külön szubrutin hajtja végre:

```
1010 M$="                    "
1020 G$="          ":E$="    "
1030 PRINT#15,"P"CHR$(2)CHR$(RL)CHR$(RH)CHR$(P)
1040 IF P=1 THEN INPUT#2,T$
1050 IF P=3 THEN INPUT#2,M$,G$,E$
1999 RETURN
```

Ugyancsak szubrutin végzi a rekord tartalmának a megjelenítését a képernyőn:

```
2010 PRINT"■"
2020 PRINT"■SORSZAM      =";N
2030 PRINT"■ALLAPOT      = ";T$
2040 PRINT"■MEGNEVEZES = ";M$
2050 PRINT"■GYARI SZAM = ";G$
2060 PRINT"■ERTEK        = ";E$
2999 RETURN
```

Hasonló szubrutinokat a karbantartó programban már láttunk.
Gépeljük be, majd mentsük ki lemezre a programot, RELVALOGATAS néven. Ezután indítsuk el.
Először válogassunk a gyári szám szerint. Keressük azt az eszközt, amelynek a gyári száma 44–3333:

```
VALOGATAS AZ ESZKOZOK ALLOMANYBAN
---------------------------------------

<1> MEGNEVEZES SZERINT
<2> GYARI SZAM SZERINT
<3> ERTEK      SZERINT
<0> --- EGYIK  SEM ---
---------------------------------------
? MELYIK =/1/2/3/0/= ? 2
---------------------------------------
? ADAT = ? 44-3333
```

Ilyen rekord csak egy van:

```
SORSZAM     = 3
ALLAPOT     = +
MEGNEVEZES = 1541 LEMEZEGYSEG
GYARI SZAM = 44-3333
ERTEK       = 30000
---------------------------------------

TOVABBLEPES BARMELY GOMBBAL!
```

Majd válogassunk a megnevezés szerint. Keressük azokat az eszközöket, amelyeknek a megnevezése C64-gyel kezdődik:

```
VALOGATAS AZ ESZKOZOK ALLOMANYBAN
---------------------------------------

<1> MEGNEVEZES SZERINT
<2> GYARI SZAM SZERINT
<3> ERTEK      SZERINT
<0> --- EGYIK  SEM ---
---------------------------------------
? MELYIK =/1/2/3/0/= ? 1
---------------------------------------

? ADAT = ? C64
```

Ebből már kettőt talál a program:

```
SORSZAM     = 1
ALLAPOT     = +
MEGNEVEZES = C64 KOZP.EGYS.
GYARI SZAM = 44-1111
ERTEK       = 20000
---------------------------------------

TOVABBLEPES BARMELY GOMBBAL!


SORSZAM     = 2
ALLAPOT     = +
MEGNEVEZES = C64 TAPEGYSEG
GYARI SZAM = 44-2222
ERTEK       = 5000
---------------------------------------

TOVABBLEPES BARMELY GOMBBAL!
```

Végül válogassunk érték szerint. Keressük azokat az eszközöket, amelyeknek értéke 50000 forint:

```
VALOGATAS AZ ESZKOZOK ALLOMANYBAN
-------------------------------------

<1> MEGNEVEZES SZERINT
<2> GYARI SZAM SZERINT
<3> ERTEK      SZERINT
<0> --- EGYIK  SEM ---
-------------------------------------
? MELYIK =/1/2/3/0/= ? 3
-------------------------------------
? ADAT = ? 50000
```

Ha a megadott adatokkal töltöttük fel az állományt, ilyen eszközt a program nem fog találni.

Most már (0 válasszal) leállíthatjuk a programot.

Akármilyen jól működött is a program, nem lehetünk vele elégedettek, mert eléggé primitív, csak a válogatás alapvető menetének illusztrálására alkalmas.

```
100 REM-------------------------------MEGNYITASOK
110 PRINT"□"
120 OPEN 15,8,15:OPEN 2,8,2,"ESZKOZOK"
130 K$="-------------------------------------"
140 NN=20
200 REM-------------------------------ADATBEKERES
210 PRINT"□"
212 :   PRINT"VALOGATAS AZ ESZKOZOK ALLOMANYBAN"
214 :   PRINT K$
220 :   PRINT"◙<1> MEGNEVEZES SZERINT"
222 :   PRINT"◙<2> GYARI SZAM SZERINT"
224 :   PRINT"◙<3> ERTEK      SZERINT"
226 :   PRINT"◙<0> --- EGYIK  SEM ---"
228 :   PRINT K$
230 :   PRINT"◙? MELYIK =/1/2/3/0/= ";
232 :   INPUT W$:W=VAL(RIGHT$(W$,1))
234 IF W<0 OR W>3 THEN GOTO 210
236 IF W=0        THEN GOTO 910
240 :   PRINT K$
242 :   INPUT"◙? ADAT =    -████";A$
244 IF A$="-"     THEN GOTO 210
246 :   L=LEN(A$):PRINT"□"
300 REM---------------------------------VALOGATAS
310 :   FOR N=1 TO NN
320 :   :   RH=INT(N/256):RL=N-RH*256
330 :   :   P=1:GOSUB 1010
```

```
340 :  :  IF T$<>"π" THEN P=3:GOSUB 1010
350 :  :  GOSUB 2010
410 :  :  ON W GOTO 430,440,450
420 :  GOTO 599
430 :  :  X$=M$:GOTO 460
440 :  :  X$=G$:GOTO 460
450 :  :  X$=E$:GOTO 460
460 :  :  X$=LEFT$(X$,L)
470 :  IF X$<>A$ THEN GOTO 599
510 :  :  PRINT"▮";K$
520 :  :  PRINT"▮TOVABBLEPES BARMELY GOMBBAL!"
530 :  :  GET B$:IF B$="" THEN GOTO 530
599 :  NEXT N
699 GOTO 210
900 REM-----------------------------------LEZARASOK
910 PRINT"▯"
920 CLOSE 2:CLOSE 15
999 END
1000 REM=====================================OLVASAS
1010 M$="                    "
1020 G$="          ":E$="      "
1030 PRINT#15,"P"CHR$(2)CHR$(RL)CHR$(RH)CHR$(P)
1040 IF P=1 THEN INPUT#2,T$
1050 IF P=3 THEN INPUT#2,M$,G$,E$
1999 RETURN
2000 REM================================MEGJELENITES
2010 PRINT"▩"
2020 PRINT"▮SORSZAM     =" ;N
2030 PRINT"▮ALLAPOT     = ";T$
2040 PRINT"▮MEGNEVEZES  = ";M$
2050 PRINT"▮GYARI SZAM  = ";G$
2060 PRINT"▮ERTEK       = ";E$
2999 RETURN
```

Természetesen nincs akadálya, hogy a programot ízlésünk szerint csinosítsuk. Sőt, javasoljuk, hogy ezt tegyük is meg. A teljesség igénye nélkül néhány tippet adunk:

A program nem használja a parancscsatornát, így hibafigyelést sem végez, pedig célszerű lenne, legalább a megnyitás és a pozicionálások után. Ha már figyeljük a parancscsatornát, figyelhetnénk az állomány végét is, a H = 50 hibakódot. Jelenleg a program meghatározott NN számú rekordra hajtja végre a válogatást. Bekérhetné ezt az adatot a billentyűzetről is, sőt akár intervallumot is kérhetne. Most már a válogatást végrehajthatnánk akár az állomány elejétől egy adott rekordig, akár egy adott rekordtól az állomány végéig, vagy két adott rekord között; ami tulajdonképpen részleges válogatást jelentene.

A program a törölt rekordokat is válogatja, pedig az állapotjelző figyelésével kihagyhatná őket a válogatásból. Ugyanez a helyzet a feltöltetlen rekordokkal. Ez nem hátrány abból a szempontból, hogy ezeket a rekordokat is megjeleníti, így a válogatás előrehaladása

nyomon követhető. Persze ezt megtehetné úgy is, hogy a válogatási szempontot ilyenkor nem ellenőrzi, hiszen az úgysem teljesül.

Egyébként ha tudjuk, hogy az állomány folyamatosan van feltöltve, mert például a felvivő eljárás mást nem tesz lehetővé, akkor feltöltetlen rekordok csak az állomány végén lehetnek. Ez esetben az első feltöltetlen rekord elérésekor leállíthatjuk a válogatást.

A válogatási szempont teljesülésének figyelésekor a program csak azt ellenőrzi, hogy a mező az adott A$ adattal kezdődik-e. Könnyen megoldható lenne a figyelés úgy is, hogy az adott A$ jelsorozatnak az adott mező tetszőleges helyén való előfordulását ellenőrizze a program. Ez jóval szélesebb körű válogatást tenne lehetővé; így például kiválogathatnánk azokat az eszközöket, amelyeknek a megnevezésében szerepel a "NYOMTATO" jelsorozat.

Tovább finomíthatjuk a válogatást, ha a fenti figyelést csak a szöveges mezőkre engedjük meg, a numerikus mezőknek nem a karaktersorozatát, hanem a numerikus értékét figyeljük. Ez esetben nemcsak az =, hanem a < és > relációknak is van értelme a válogatási feltételben. Így például kiválogathatnánk azokat az eszközöket, amelyeknek értéke 5000 Ft felett van.

Programunk csak képernyőre válogat. Nem okozna nagy gondot az sem, hogy a kiválogatott eszközöket nyomtatóra is kilistázza, akár úgy, hogy ez a lista a teljes rekordot tartalmazza, akár úgy, hogy csak az eszköz azonosításához szükséges adatokat írja ki. Még jobb, ha a program mindkettőt tudja, és a gépkezelő döntheti el, hogy hová kéri a válogatást. Még nem túl bonyolult egy olyan válogatás sem, amely automatikusan a képernyőre válogat, és a gépkezelő a kiválogatott rekordok ismeretében utólag dönthet, hogy kér-e róluk nyomtatott listát, avagy sem. Ilyenkor az is megoldható, hogy csak a gépkezelő által kiválasztott rekordokat tartalmazza a lista.

Eddig mindig csak egy adott mező szerint tudott programunk válogatni. Valamivel bonyolultabb, ha egyidejűleg több, akár az összes mező szerint végrehajtható a válogatás. Ez akkor a legegyszerűbb, ha az egyes mezőkre kirótt válogatási feltételek között csak logikai "ÉS" (AND) kapcsolat áll fenn. Így például kiválogathatók azok az eszközök, amelyeknek a megnevezése C64-gyel, a gyári száma pedig 44-gyel kezdődik, és az értékük nagyobb mint 10000 forint. Nem jelent nehézséget az sem, hogyha több, de nem az összes mező szerint akarunk válogatni; ilyenkor a vizsgálatból kihagyott mező tartalmát egyszerűen nem figyeljük. Ugyanez a helyzet, ha az egyes feltételek között kizárólag logikai "VAGY" (OR) kapcsolat áll fenn, ámbár az ilyen válogatásnak kisebb a gyakorlati jelentősége.

Érdekesebb, és gyakorlati szempontból nagyobb jelentőségű az a válogatás, amelyben az egyes mezőkre kirótt feltételek között AND és OR kapcsolat vegyesen is előfordulhat. Ennek megvalósítása azonban már meglehetősen bonyolult, hiszen nemcsak a mezőt, valamint az abban előforduló adatot kell bekérni és figyelni, hanem a feltételek közötti logikai kapcsolatot is, és a figyelést ezeknek megfelelően kell megszervezni, illetve végrehajtani.

Még ennél is bonyolultabb, ha mindezen felül az egyes mezőkre összetett feltételeket is megengedünk. Például összetett feltétellel válogathatók ki azok az eszközök, amelyeknek az értéke legalább 10000, de legfeljebb 40000 forint. (Itt a feltétel: E>=10000 AND E<=40000.) Hasonlóképpen kiválogathatók azok az eszközök, amelyeknek a megnevezésében vagy a "KOZPONTI", vagy a "LEMEZ", vagy a "NYOMTATO" szövegrész szerepel.

Az összetett feltételes válogatás persze csak akkor bonyolult, ha teljesen általánosan akarjuk megoldani; azaz minden feltétel, valamint a köztük levő logikai kapcsolatok megadását a gépkezelőre bízzuk, aki teljesen szabadon választhat közülük bármilyen kombinációt. A gyakorlatban erre azonban ritkán van szükség, hiszen sok kombináció egész egyszerűen értelmetlen, sőt még az elvileg értelmes válogatási szempontok nagy részére sincs gyakorlatilag szükség. A kombinált válogatás tehát lényegesen leegyszerűsíthető, ha a válogatási szempontok egy előre megtervezett készletéből lehet csak választani.

Bizonyos esetekben további egyszerűsítést eredményezhet a többmenetes válogatás alkalmazása. Ez azt jelenti, hogy a válogatóprogram az adott szempont szerint kiválogatott rekordokból egy új állományt hoz létre, amely azután ugyanazzal a válogatóprogrammal egy másik szempont szerint tovább válogatható, a szempontokat megfelelően cserélve akár többször is. Ilyenkor kell egy külön listázóprogram, amely a végső lemezállományt képernyőn vagy nyomtatón megjeleníti. A válogatott állomány persze már lehet egyszerű soros állomány is. Különbséget kell tenni az elsődleges válogatás és a másodlagos válogatások között; az előbbi az eredeti relatív állományon hajtódik végre, az utóbbiak már a válogatott soros állományokon. Esetenként gyorsabb, de mindenképpen helytakarékosabb a válogatás, ha a válogatott soros állomány nem magukat az eredeti relatív állományból kiválogatott rekordokat, hanem ezeknek csak a rekordsorszámát tartalmazza.

A leghatásosabb egyszerűsítési mód azonban az átgondolt és előrelátó adatszervezés. Ha a rekordot a lehetséges válogatási szempontok figyelembevételével építjük fel, és a mezőket, illetve az adataikat ennek megfelelően állítjuk össze, igen egyszerű válogatóprogrammal is kielégíthetjük az igényeket.

Tekintettel arra, hogy az igények általában nem egyidejűleg és nem egyforma gyakorisággal jelentkeznek, sokszor az a legjobb megoldás, ha nem egy mindent tudó válogatóprogramot írunk, hanem több speciálisat, amelyek mindegyike csak egy-egy meghatározott szempontrendszer szerint tud válogatni. Az ilyen programok az igények szerint egymástól függetlenül futtathatók, és mivel kisebbek és egyszerűbbek, gyorsabban betölthetők, könnyebben kezelhetők, egyszerűbben módosíthatók. (Ennek különösen a COMMODORE esetén van jelentősége, ahol a tárkapacitás szűk keresztmetszet.)

Mindezeket a válogatási lehetőségeket csak megemlítjük, de nem mutatjuk be, hiszen sorozatunk e kötetének célja végül is nem a válogatási módszerek, hanem a random és relatív állományok kezelésének bemutatása.

Mindazonáltal a bemutatott mintaprogram alkalmas a különböző válogatási lehetőségek kipróbálására, hiszen szerkezete nem az adott feladathoz, hanem a válogatás általános alapelveihez igazodik, így bármilyen válogatóprogram váza lehet. Ezért javasoljuk, hogy aki komolyan akar foglalkozni ezzel a témával, az említett módosításokat, legalábbis az egyszerűbbeket, fokozatosan építse be a mintaprogramba, az egyes változatokat alaposan tesztelje és elemezze.

LEKÉPEZÉSI ELJÁRÁSOK

Mint láthattuk, mind a random, mind a relatív állományoknál kulcskérdés volt, hogy milyen összefüggést tudunk találni vagy létesíteni a rekordok és a tárolási helyük között.

Általában szükséges, hogy minden rekordnak legyen egy azonosítója, azaz egy olyan adata, amely megkülönbözteti a rekordot az összes többitől. Egyszerűsíti a helyzetet, ha ez az adat egyedi, azaz a rekordot egyértelműen azonosítja. Ilyen például egy dolgozó

személyi adatait tartalmazó rekordban a személyi szám. Nem ilyen viszont a dolgozó neve. Egy munkahelyen több KOVACS JANOS is dolgozhat. A név tehát nem azonosítja egyértelműen a rekordot. Sok esetben azonban meg kell elégednünk az ilyen azonosítóval is, ami természetesen a rekordok kezelését megnehezíti.

A rekord és a tárolási hely közötti összefüggés azt jelenti, hogy a rekord azonosítója alapján meghatározható egy cím a lemezen, ahol a rekord tárolható. Ez a cím lehet abszolút, mint a random állománynál, ahol egy adott sávcímből és szektorcímből áll. De lehet relatív is, mint a relatív állománynál, ahol az állomány fizikai kezdetétől számított rekordpozíciót jelenti.

Azt az eljárást, amellyel a rekord azonosítójából meghatározzuk a rekord (abszolút vagy relatív) címét, leképezési eljárásnak vagy röviden leképezésnek nevezzük. A leképezésnek több fontos tulajdonsága van. Ha fel akarunk használni egy leképező eljárást, alaposan tisztában kell lennünk ezekkel a tulajdonságokkal, hogy eldönthessük, alkalmas-e az adott feladatban szerepének betöltésére.

Javasoljuk, hogy a leképezési eljárásokat a tényleges felhasználásuk előtt a tárban próbáljuk ki és teszteljük le. Ez nem nehéz, hiszen akár a random, akár a relatív állomány lemezterületét könnyen szimulálhatjuk a tárban. A relatív állomány esetén ez egy egydimenziós tömb (vektor), amelynek minden egyes eleme egy-egy rekordot jelent, és az elemek indexei a rekordsorszámok, azaz a relatív címek:

```
RELATIV LEKEPEZES

        5    0    5    0    5
   ----+----|----+----|----+----
1  ::::::::::::::::::::::::::::::   30
   ----+----|----+----|----+----
        5    0    5    0    5
```

A random állomány esetén pedig egy kétdimenziós, 35-ször 20-as tömböt (mátrixot) kell használnunk, amelynek minden egyes eleme egy-egy blokkot jelent, és az elemek sorindexe a sávcím, oszlopindexe pedig a szektorcím. Ezek együtt adják az abszolút blokkcímet:

```
RANDOM LEKEPEZES

   01234567890123456789 0
   0----+----1----+----2

1  ::::::::::::::::::::
2  ::::::::::::::::::::
3  ::::::::::::::::::::
4  ::::::::::::::::::::
5  ::::::::::::::::::::
6  ::::::::::::::::::::
7  ::::::::::::::::::::
```

```
 8 :::::::::::::::::::::
 9 :::::::::::::::::::::
10 :::::::::::::::::::::
11 :::::::::::::::::::::
12 :::::::::::::::::::::
13 :::::::::::::::::::::
14 :::::::::::::::::::::
15 :::::::::::::::::::::
16 :::::::::::::::::::::
17 :::::::::::::::::::::
18 ---------------------
19 :::::::::::::::::::--
20 :::::::::::::::::::--
21 :::::::::::::::::::--
22 :::::::::::::::::::--
23 :::::::::::::::::::--
24 :::::::::::::::::::--
25 ::::::::::::::::::---
26 ::::::::::::::::::---
27 ::::::::::::::::::---
28 ::::::::::::::::::---
29 ::::::::::::::::::---
30 ::::::::::::::::::---
31 :::::::::::::::::----
32 :::::::::::::::::----
33 :::::::::::::::::----
34 :::::::::::::::::----
35 :::::::::::::::::----

   0----+----1----+----2
   012345678901234567890
```

Itt kissé bonyolítja a helyzetet, hogy a tömbnek nem mindegyik eleme használható, hiszen nem mindegyik tömbelemnek felel meg létező, illetve a random állomány számára felhasználható blokk. (Ezeket "–" jellel jeleztük.) Ugyanis 0-s sáv nincs, a belső sávokon kevesebb szektor van, a 18-as sávot pedig a lemezkezelő a lemeznyilvántartáshoz használja.

Célszerű olyan tesztprogramokat írni, amelyek a relatív, illetve random állományok helyett ilyen tömböket definiálnak; alkalmasak arra, hogy bármilyen adott (billentyűzetről begépelt) rekordazonosítót leképezzenek a tömbre, azaz meghatározzák a rekord címét (indexét), és a megfelelő tömbelembe egy jelet tegyenek, majd a gépkezelő kérésére a tömböt megjelenítsék. A leképező eljárást cserélhető szubrutinként kell tartalmazniuk. Relatív állományok esetén ez a tesztprogram például ilyen lehet:

```
100 REM-----------------------------------------BEJELENTKEZES
110 PRINT"♥"
120 PRINT" RELATIV LEKEPEZES"
130 PRINT" -------------------"
190 E=3:NN=50
199 DIM RE$(NN)
200 REM--------------------------------------------BEALLITAS
210 FOR I=1 TO NN
220 :  RE$(I)=":"
230 NEXT I
300 REM--------------------------------------------FELTOLTES
310 INPUT"▣ REKORD=";R$
320 IF R$="-" THEN GOTO 410
330 :  GOSUB 1010 :REM==> LEKEPEZES
340 IF R=0 THEN GOTO 310
350 :  PRINT"♥";TAB(25);"CIME=";R
360 :  RE$(R)="♦"
399 GOTO 310
400 REM----------------------------------------MEGJELENITES
410 PRINT"♥"
420 OPEN 4,E
425 PRINT#4,"RELATIV LEKEPEZES":PRINT#4
426 PRINT#4,"           5    0    5    0    5   "
430 PRINT#4,"      ┌┬┬┬┼┬┬┬┬|┬┬┬┬┼┬┬┬┬|┬┬┬┬┼┬┬┬┬┐"
436 PRINT#4
440 FOR I=1 TO NN STEP 30
450 :  PRINT#4,RIGHT$("  "+STR$(I),3)+" ";
460 :  K=I+30-1
470 :  IF K>NN THEN K=NN
480 :  FOR J=I TO K
490 :  :  PRINT#4,RE$(J);
499 :  NEXT J
510 :  PRINT#4,SPC(30-K+I);RIGHT$("  "+STR$(K),3)
599 NEXT I
610 PRINT#4
615 PRINT#4,"      └┴┴┴┼┴┴┴┴|┴┴┴┴┼┴┴┴┴|┴┴┴┴┼┬┬┬┬┘"
616 PRINT#4,"           5    0    5    0    5   "
620 CLOSE 4
999 END
1000 REM====================================LEKEPEZES
1010 R=0
1999 RETURN
```

110–130: A program bejelentkezik.

190 : Definiálja a relatív állomány rekordjainak NN számát, valamint a megjelenítésre használt eszköz E egységszámát.

199 : Definiálja a relatív állományt szimuláló RE$ tömböt.

210–230: Feltölti ezt a tömböt egy egyezményes jellel.

310–399: Bekéri a rekord R$ azonosítóját, leképezi az állományra (tömbre), megadja a rekord R címét (indexét), bejegyzi az RE$(R) rekordba (tömbelembe), hogy azt már leképezte, majd újabb R$ rekordazonosítót kér. Ha "–" leállítójelet kap, áttér a tömb kiírására.

410–999: Megjeleníti az RE$ tömböt a megadott E egységen, majd leáll. Hogy a képernyőre is kiférjen, a tömböt 30 rekordos (elemű) szakaszokra bontja. Minden szakasz elején és végén megadja a szakasz első, illetve utolsó rekordjának (elemének) a rekordcímét (indexét). A többi cím (index) ábráról való leolvasását megkönnyíti az ötösével és tízesével tagolt fejléc és lábléc.

1010–1999: Ide kell kódolni a leképező eljárást, amelynek bemenő paramétere az R$ rekordazonosító, kimenő paramétere pedig az R rekordcím (index). Sikertelen leképezés esetén ennek értéke nulla.

Megjegyezzük, hogy az állomány méretét és a kiíró (output) eszközt a 190-es sorban szükség szerint megváltoztathatjuk.

Random állományok esetén a program teljesen hasonló szerkezetű; csupán a tömb kezelésében tér el:

```
100 REM-----------------------------------BEJELENTKEZES
110 PRINT"□"
120 PRINT" RANDOM LEKEPEZES"
130 PRINT" ----------------"
190 E=3
199 DIM RA$(35,20)
200 REM---------------------------------------BEALLITAS
210 FOR I=1 TO 35:FOR J=0 TO 20
220 :   RA$(I,J)=":"
230 NEXT J:NEXT I
240 FOR J=0  TO 20:RA$(18,J)="-":NEXT J
250 FOR I=19 TO 35:RA$(I,20)="-":NEXT I
260 FOR I=19 TO 35:RA$(I,19)="-":NEXT I
270 FOR I=25 TO 35:RA$(I,18)="-":NEXT I
280 FOR I=31 TO 35:RA$(I,17)="-":NEXT I
300 REM---------------------------------------FELTOLTES
310 INPUT"■ REKORD=";R$
320 IF R$="-" THEN GOTO 410
330 :   GOSUB 1010 :REM==> LEKEPEZES
340 IF T=0 AND B=0 THEN GOTO 310
350 :   PRINT"□";TAB(20);"CIME=";T;",";B
360 :   RA$(T,B)="♦"
399 GOTO 310
400 REM------------------------------------MEGJELENITES
410 PRINT"□"
420 OPEN 4,E
425 PRINT#4,"RANDOM LEKEPEZES"
```

```
426 PRINT#4
430 PRINT#4,"     012345678901234567890"
435 PRINT#4,"     0┬┬┬┬+┬┬┬┬1┬┬┬┬+┬┬┬┬2"
436 PRINT#4
440 FOR I=1 TO 35
450 :  PRINT#4,RIGHT$("   "+STR$(I),3)+"  ";
460 :  IF I<>18 OR E>3   THEN GOTO 480
470 : :  GET B$:IF B$="" THEN GOTO 470
480 :  FOR J=0 TO 20
490 : :  PRINT#4,RA$(I,J);
499 :  NEXT J
510 :  PRINT#4
599 NEXT I
610 PRINT#4
614 PRINT#4,"     0┴┴┴┴+┴┴┴┴1┴┴┴┴+┴┴┴┴2"
615 PRINT#4,"     012345678901234567890"
620 CLOSE 4
999 END
1000 REM=================================LEKEPEZES
1010 T=0:B=0
1999 RETURN
```

110–130: A program bejelentkezik.

190: Definiálja a megjelenítésre használt eszköz E egységszámát. (Az NN állományméretre itt nincs szükség, hiszen ez a lemez szerkezetéből adódik.)

199: Definiálja a random állományt szimuláló RA$ tömböt 35 sávra, sávonként 21 szektorra.

210–280: Feltölti ezt a tömböt egyezményes jelekkel. A nem használható blokkokat (tömbelemeket) eltérő jellel írja felül.

310–399: Bekéri a rekord R$ azonosítóját, és leképezi az állományra (tömbre). Megadja a rekord címét T sávcím (sorindex) és B szektorcím (oszlopindex) formájában. Bejegyzi az RA$(T,B) rekordba (tömbelembe), hogy már leképezte, majd újabb R$ rekordazonosítót kér. Ha "–" leállítójelet kap, áttér a tömb kiírására.

410–999: Megjeleníti az RA$ tömböt a megadott E egységen, majd leáll. A tömböt sávonkénti bontásban (21 elemenként) írja ki. Minden sávhoz oldallécben megadja a sávcímet (sorindexet). A szektorcímeket (oszlopindexeket) a fejlécben és a láblécben tünteti fel. Ha képernyőre ír, a 18. sáv előtt megáll, hogy a lista ne futhasson ki a képernyőről. A kiírás bármely gomb megnyomására folytatódik.

1010–1999: Ide kell kódolni a leképező eljárást, amelynek a bemenő paramétere az R$ rekordazonosító, a kimenő paraméterei pedig a T sávcím (sorindex) és a B szektorcím (oszlopindex). Sikertelen leképezés esetén ezek értéke nulla.

Tulajdonképpen a random állományok leképezésére nincs igazán szükségünk, mivel az esetek többségében az indextábla megfelelő megoldást jelent, és többnyire jobb is, mint

az általában igen bonyolult leképező eljárás. A programot egyrészt a teljesség kedvéért mutatjuk be, másrészt bizonyos leképezési sajátosságok illusztrálására.

Ezek közül az első a címtartomány, ami nem más, mint a megengedett címek halmaza. A lemez és az állományszervezési módok fizikai sajátosságai miatt ugyanis a lemezkezelő csak bizonyos címeket fogad el. A random állományok esetén a rekordcím csak létező blokk abszolút címe lehet. Relatív állomány esetén a relatív cím, azaz a rekordsorszám csak 0-nál nem kisebb és 65535-nél nem nagyobb egész szám lehet.

A címtartományt tovább szűkíthetik egyéb járulékos feltételek. Ilyen például a relatív állományoknál a rekord hossza, mivel az állomány nem foglalhat el 167132 bájtnál nagyobb területet a lemezen; így mondjuk 200 bájtos rekordok esetén a címtartomány felső határa 835. Random állományoknál például a 18-as sáv az ott tárolt lemeznyilvántartás miatt nem használható, noha a címei létező blokkokat határoznak meg, ezért a random állományhoz legfeljebb 664 blokk tartozhat. A címtartomány tehát nem az elméletileg lehetséges, hanem az adott állomány esetén gyakorlatilag használható címeket foglalja magában.

A relatív állomány címtartománya mindig folytonos; az első rekord címétől az utolsó rekord címéig tart. A random állomány címtartománya ezzel szemben szakaszos (diszkrét), a lemez használható blokkjainak címére terjed ki, az (1,0)-s blokktól az (1,20)-asig, majd a (2,0)-stól a (2,20)-asig, és így tovább, a (35,16)-os blokkig, kihagyva közben a nem létező, illetve nem használható blokkokat. A leképező eljárás tehát csak akkor használható, ha az aktuális címtartományra képez le.

Tekintsünk egy olyan random állományt, amelynek a rekordjait a dátum azonosítja, pontosabban a hónap és a nap.

Ha most olyan leképező eljárást választunk, amely a hónapból sávcímet, a napból szektorcímet képez, akkor címtartományon kívüli címeket is elő fog állítani:

jan. 1.	azonosító:	0101	cím:	1. sáv	1. szektor
nov. 18.	azonosító:	1118	cím:	11. sáv	18. szektor
feb. 26.	azonosító:	0226	cím:	2. sáv	26. szektor

Ezek közül az első kettő még létező blokkcím, de az utolsó már nem az. Sőt nem ez az egyetlen; többek között minden hónap 20. napját követő napok rekordjai ilyenek lesznek.

Ha a leképezést úgy választjuk meg, hogy a napból képezze a sávcímet és a hónapból a szektorcímet, akkor ez a hiba elkerülhető:

jan. 1.	azonosító:	0101	cím:	1. sáv	1. szektor
nov. 18.	azonosító:	1118	cím:	18. sáv	11. szektor
feb. 26.	azonosító:	0226	cím:	26. sáv	2. szektor

Most azonban új probléma áll elő; a 18-as sávra is leképződnek rekordok, márpedig ott a lemeztérkép és a tartalomjegyzék van. Úgy kell tehát módosítanunk a leképezést, hogy a 18-as sávra kerülő rekordokat a 32-es sávra képezze le:

```
1010 T=0:B=0
1020 H=INT(VAL(LEFT$ (R$,2)))
1030 N=INT(VAL(RIGHT$(R$,2)))
1040 IF H<1 OR H>12 THEN GOTO 1999
```

```
1050 IF N<1 OR N>31 THEN GOTO 1999
1060 T=N:B=H
1070 IF T=18 THEN T=32
1999 RETURN
```

Építsük be a random leképezéshez készített programba, és futtassuk le az előbbi adatokra.

```
RANDOM LEKEPEZES
----------------

REKORD=? 0101        CIME= 1 , 1
REKORD=? 1118        CIME= 32 , 11
REKORD=? 0226        CIME= 26 , 2
REKORD=? -
```

```
RANDOM LEKEPEZES

    012345678901234567890
    0┬┬┬┬+┬┬┬┬1┬┬┬┬+┬┬┬┬2

 1  :◆:::::::::::::::::::
 2  :::::::::::::::::::::
 3  :::::::::::::::::::::
 4  :::::::::::::::::::::
 5  :::::::::::::::::::::
 6  :::::::::::::::::::::
 7  :::::::::::::::::::::
 8  :::::::::::::::::::::
 9  :::::::::::::::::::::
10  :::::::::::::::::::::
11  :::::::::::::::::::::
12  :::::::::::::::::::::
13  :::::::::::::::::::::
14  :::::::::::::::::::::
15  :::::::::::::::::::::
16  :::::::::::::::::::::
17  :::::::::::::::::::::
18  ---------------------
19  :::::::::::::::::::--
20  :::::::::::::::::::--
21  :::::::::::::::::::--
22  :::::::::::::::::::--
23  :::::::::::::::::::--
24  :::::::::::::::::::--
25  ::::::::::::::::::---
```

```
26 ::◆:::::::::::::::---
27 ::::::::::::::::::---
28 ::::::::::::::::::---
29 ::::::::::::::::::---
30 ::::::::::::::::::---
31 :::::::::::::::::----
32 :::::::::::◆:::::----
33 :::::::::::::::::----
34 :::::::::::::::::----
35 :::::::::::::::::----

   0┴┴┴┴┼┴┴┴┴1┴┴┴┴┼┴┴┴┴2
   012345678901234567890
```

Az eredmények azért érdekesek, mert rámutatnak a leképezési eljárások sajátosságaira. Ha a rekordazonosító változtatás nélkül meghatározza a rekordcímet, akkor a leképezés közvetlen.

A példa szerinti majdnem ilyen, hiszen a nap közvetlenül meghatározza a sávot, a hónap pedig a szektort, a 18-as sáv kivételével. Itt tehát a leképezés egy címkiszámító eljárásra épül. Az ilyen leképezést közvetettnek nevezzük.

Mivel általában meg kell akadályoznunk, hogy a képzett cím kimutasson a címtartományból, a gyakorlatban szinte alig használunk közvetlen leképezést. Különösen igaz ez a random állományok esetén, ahol a címtartomány eléggé szűk, és nem összefüggő.

Megjegyezzük, hogy az indextáblás eljárás tulajdonképpen egy speciális közvetett leképezés, amely minden rekordazonosítóhoz az első szabad blokk címét rendeli hozzá.

A leképezés igen jellemző tulajdonsága, hogy a rekordokat hogyan „teríti", azaz hogy az adott rekordazonosítókból képzett címekkel mennyire fedi le a címtartományt. Ez a tulajdonság meghatározza a leképezés gazdaságosságát abban a tekintetben, hogy menynyire használja ki a lemezterületet. Ilyen értelemben a most kipróbált leképezés nem hézagmentes. A 32. sávtól felfelé minden sávot teljesen üresen hagy, de az alacsonyabban fekvő sávokon is csak az 1-től 12-ig terjedő szektorokat használja ki.

Próbáljuk meg ugyanezeket a rekordokat egy relatív állományban tárolni. Tekintettel az eltérő címtartományra, itt más leképező eljárásra van szükségünk: a rekord címe legyen az adott hónapot megelőző hónapok napjainak, valamint az adott dátumban lévő napok számának összege. Az egyszerűség kedvéért minden hónapot 31 naposnak tekintsünk:

jan.	1.	azonosító:	0101	cím:	1	(=0∗31+1)
nov.	18.	azonosító:	1118	cím:	328	(=10∗31+18)
feb.	2.	azonosító:	0226	cím:	57	(=1∗31+26)

A címtartomány tehát 1–372 lesz. Ennek megfelelően a leképező eljárás:

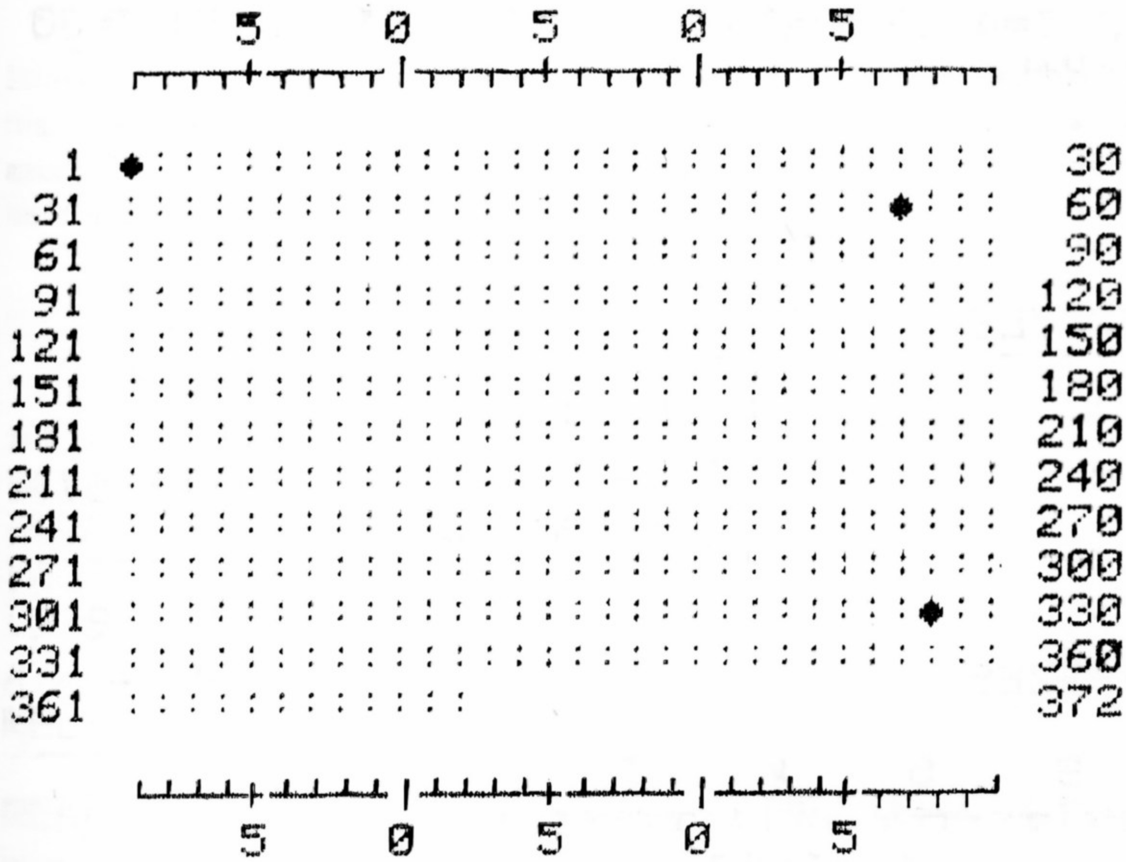
```
1010 R=0
1020 H=INT(VAL(LEFT$ (R$,2)))
1030 N=INT(VAL(RIGHT$(R$,2)))
1040 IF H<1 OR H>12 THEN GOTO 1999
1050 IF N<1 OR N>31 THEN GOTO 1999
1060 R=(H-1)*31+N
1999 RETURN
```

Ezt gépeljük hozzá a relatív leképezést tesztelő programunkhoz, és próbáljuk ki a fenti adatokkal.

FIGYELEM!
Ne felejtsük el a 190-es sorban az állomány méretét NN=372 értékre módosítani!

Ha megnézzük a leképezés eredményét az állomány teljes feltöltöttsége esetén, láthatjuk, hogy ez a leképezés sem hézagmentes:

RELATIV LEKEPEZES



A 31 napnál rövidebb hónapok „hiányzó" napjainak megfelelő rekordok helye mindig szabad marad. E leképezés másik jellemzője, hogy egyértelmű. Ez azt jelenti, hogy két különböző rekordazonosítót soha nem képez le ugyanarra a címre. Ennél erősebb megszorításnak is eleget tesz: egy adott rekordazonosítót mindig ugyanarra a címre képez le. További előnye, hogy kis ráfordítással hézagmentessé is tehető: a hónapokat ne egységesen 31 nappal, hanem a napjaik tényleges számával vegyük figyelembe, azaz ahányadik napja az évnek a dátum által meghatározott nap, annyi lesz rekordjának címe is. (Az egyszerűség kedvéért tekintsünk el a szökőévektől.) Ekkor a címtartomány is rövidebb lesz; csak 1-től 365-ig tart.

jan. 1.	azonosító:	0101	cím:	1
nov. 18.	azonosító:	1118	cím:	322
feb. 26.	azonosító:	0226	cím:	57

A megfelelő eljárás listája a 136. oldalon látható.

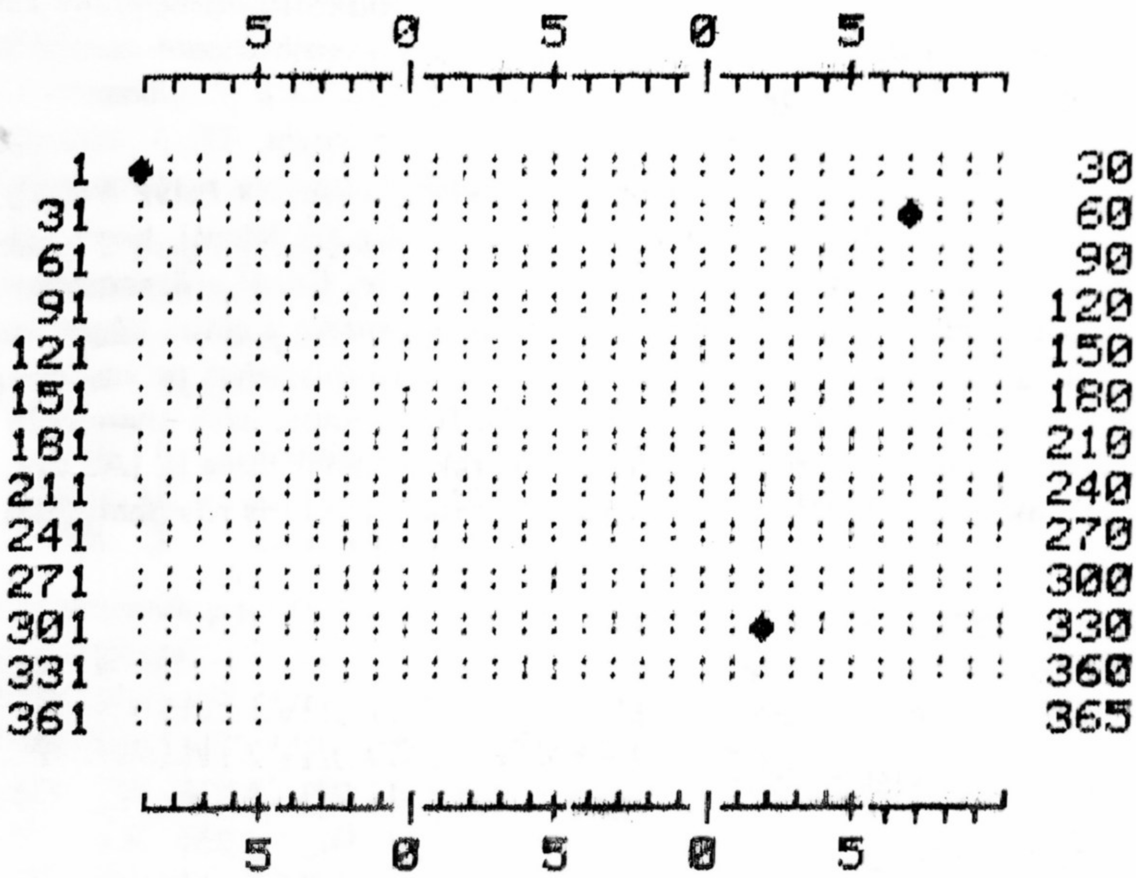
Ezt is kipróbálhatjuk, NN=365 elemű állománnyal.

```
1010 R=0
1020 H=INT(VAL(LEFT$ (R$,2)))
1030 N=INT(VAL(RIGHT$(R$,2)))
1040 IF H<1 OR H>12 THEN GOTO 1999
1050 L=31:IF H=2 THEN L=28
1060 IF H=4 OR H=6 OR H=9 OR H=11 THEN L=30
1070 IF N<1 OR N>L  THEN GOTO 1999
1110 :   R=0:IF H=1  THEN GOTO 1210
1120 :   FOR I=1 TO H-1
1130 :   :  L=31:IF I=2 THEN L=28
1140 :   :  IF I=4 OR I=6 OR I=9 OR I=11 THEN L=30
1150 :   :  R=R+L
1160 :   NEXT I
1210 R=R+N
1999 RETURN
```

```
RELATIV LEKEPEZES
-----------------

REKORD=? 0101                 CIME= 1
REKORD=? 1118                 CIME= 322
REKORD=? 0226                 CIME= 57
REKORD=? -

RELATIV LEKEPEZES
```



Ez a leképezés tehát nemcsak egyértelmű, de hézagmentes is. Tulajdonképpen mindig ez lenne az ideális. Sajnos hátránya, hogy viszonylag ritkán sikerül ilyet találni. A leképező eljárást ugyanis nemcsak a címtartomány befolyásolja, hanem a lehetséges rekordazonosítók halmaza is.

Vegyünk egy másik példát. Az azonosító most ne numerikus, hanem szöveges legyen, mondjuk minden esetben egy betűvel kezdődő név.

Legyen a leképezési eljárás az, hogy a név első betűjének ASCII kódját vesszük:

azonosító:	ALFA	cím:	65
azonosító:	BETA	cím:	66
azonosító:	GAMMA	cím:	71

Mivel a relatív állománynál mindig 1-től indul a címtartomány, jócskán hézagos lesz, hiszen az 1—90 terjedelmű tartományból csak a 65—90 részt használjuk ki. A leképezés azonban könnyen hézagmentessé tehető, és a címtartomány 1—26 terjedelműre csökkenthető, ha az ASCII kódból levonunk 64-et:

azonosító:	ALFA	cím:	1	(=65—64)
azonosító:	BETA	cím:	2	(=66—64)
azonosító:	GAMMA	cím:	7	(=71—64)

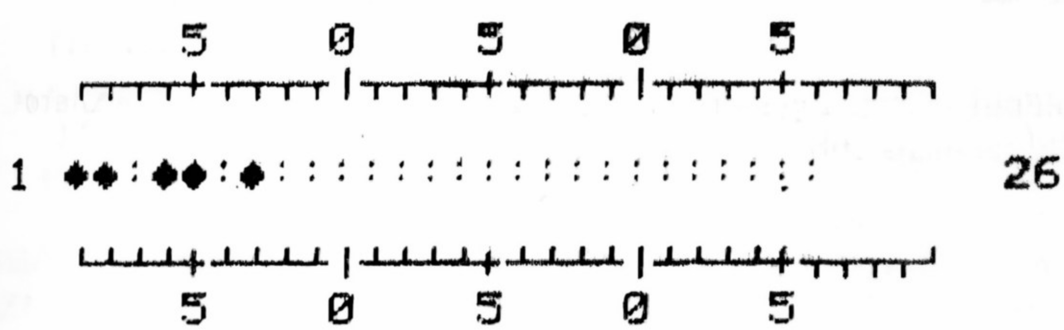
A leképező eljárás ekkor:

```
1010 R=0
1020 L=ASC(R$+"0")-64
1030 IF L>=1 AND L<=26 THEN R=L
1999 RETURN
```

A hatása pedig:

```
RELATIV LEKEPEZES
-----------------

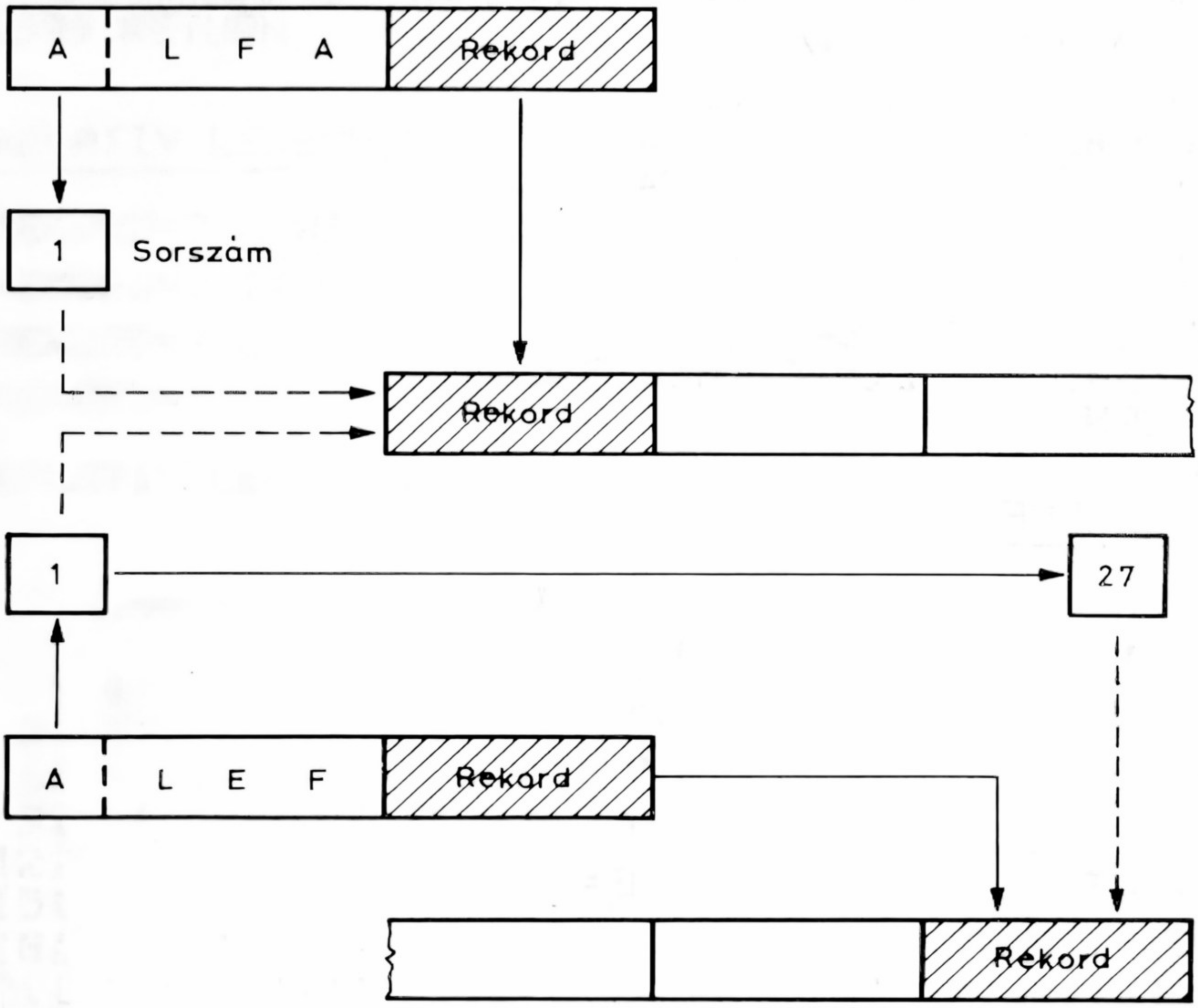
REKORD=? A                  CIME= 1
REKORD=? B                  CIME= 2
REKORD=? G                  CIME= 7
REKORD=? D                  CIME= 4
REKORD=? EP                 CIME= 5
REKORD=? ET                 CIME= 5
REKORD=? -
RELATIV LEKEPEZES
```



Látható, hogy ez a leképezés egyszerű, hézagmentes, de nem egyértelmű. Az EPSZILON és az ETA rekordazonosítókat ugyanarra a címre képezi le. Ezért ezeket szinonimáknak nevezzük. A szinonimák tulajdonképpen kétféle módon keletkezhetnek: vagy a leképező eljárás állítja őket elő, vagy már eleve előfordulnak az azonosítók között. Az előbbire példa a fenti EPSZILON–ETA pár, az utóbbira pedig az a személyi nyilvántartás, amelyben két KOVACS JANOS szerepel.

A szinonimák elleni küzdelemnek két útja van: a megsemmisítés és a békés egymás mellett élés. Az előbbi azt jelenti, hogy olyan leképező eljárást keresünk, vagy dolgozunk ki, amely nem fogad el, és nem is állít elő szinonimákat. Az utóbbi esetén pedig beletörődünk abba, hogy szinonimák vannak, és a leképező eljáráson belül (vagy ritkábban kívül) gondoskodunk a megfelelő kezelésükről.

Noha tökéletesnek a radikális megoldás tűnik, sok esetben egyszerűbb a szinonimákat kezelni, mint egy teljesen egyértelmű leképezési eljárást konstruálni.



35. ábra. Szinonímák kezelése

Vegyük például az előbbi esetet. Egyszerűen duplázzuk meg az állomány területét, és a szinonimát tegyük 26-tal magasabb címre (35. ábra).

azonosító:	ALFA	cím:	1
azonosító:	BETA	cím:	2
azonosító:	GAMMA	cím:	7

azonosító: DELTA cím: 4
azonosító: EPSZILON cím: 5 (=69—64)
azonosító: ETA cím: 31 (=69—64+26)

Az így megnövelt címet másodlagos címnek nevezzük, szemben az eredeti, elsődleges címmel.

Ilyenkor persze figyelnünk kell arra, hogy egy azonosító szinonima-e, avagy sem. Ezt onnan ismerjük fel, hogy az elsődleges cím foglalt-e, avagy szabad. (Azaz példánk szerint a tesztprogramban a címen található rekord tartalma rombusz vagy kettőspont.)

```
1010 R=0
1020 L=ASC(R$+"0")-64
1030 IF L<1 OR L>26 THEN GOTO 1999
1040 IF RE$(L)="♦"  THEN L=L+26
1050 R=L
1999 RETURN
```

```
RELATIV LEKEPEZES

           5    0    5    0    5
       ----+----|----+----|----+-----

   1   ♦♦:♦♦:♦:::::::::::::::::::::::   30
  31   ♦:::::::::::::::::::::           52

       ----+----|----+----|----+-----
           5    0    5    0    5
```

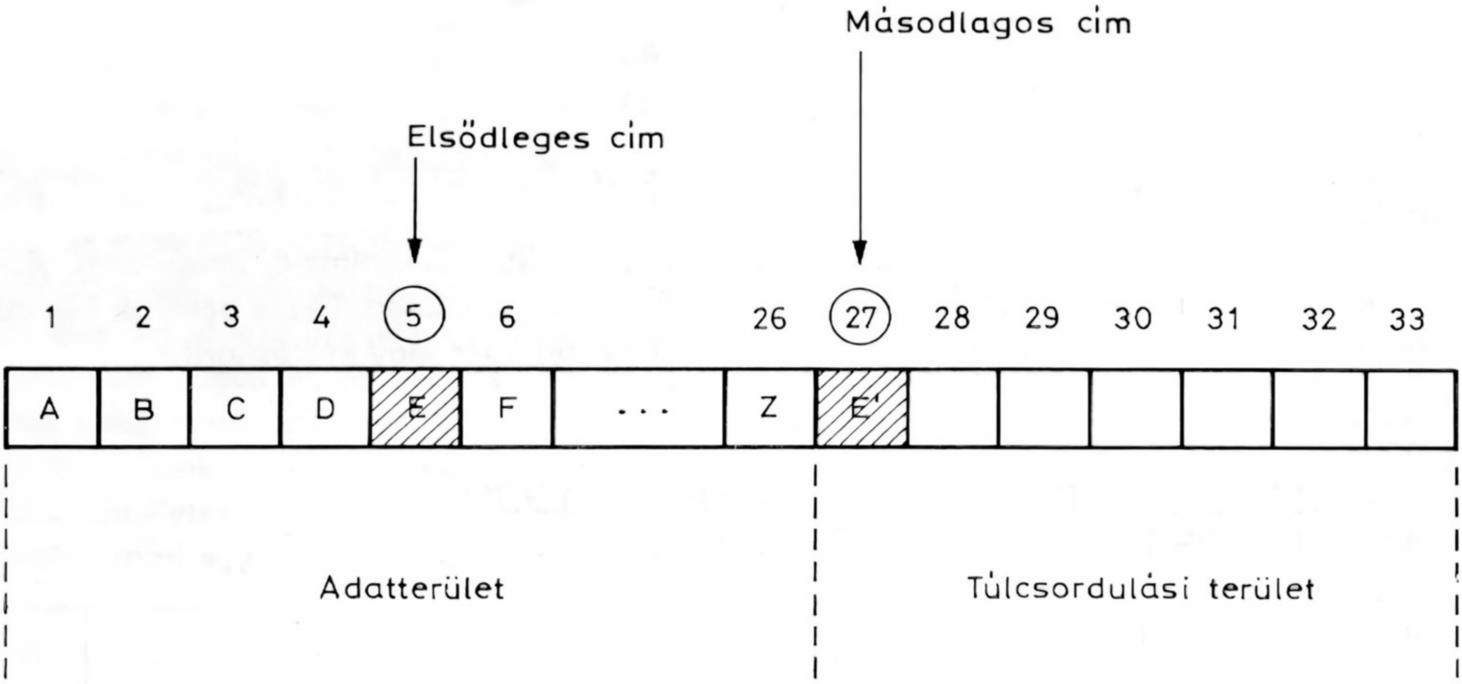
E szinonimakezelésnél persze, ha egy rekordot vissza akarunk keresni (be akarunk olvasni) az állományból, akkor azt először az elsődleges címén keressük. Ha nincs ott, mert ezen a helyen például egy szinonima van, akkor a rekordot a másodlagos címen is keresni kell. (Ebből következik, hogy a szinonimákat vagy a rekordazonosító, vagy a rekordtartalom alapján meg kell tudnunk különböztetni egymástól.)

Nyilvánvaló, hogy ez a megoldás akkor gazdaságos, ha bármely rekordnak lehet szinonimája, mégpedig ugyanolyan gyakorisággal. Viszont csak akkor működőképes, ha egy rekordnak soha nincs egynél több szinonimája.

A példaként vett görög betűk nem ilyenek. Egyrészt a szinonimával rendelkező azonosítók száma kevés, másrészt egynél több szinonimájuk is lehet:

EPSZILON	— ETA	
THETA	— TAU	
KAPPA	— KSZI	— KHI
O MIKRON	— O MEGA	
PI	— PSZI	— PHI

A hármas szinonimák „adminisztratív'' eszközökkel is megszüntethetők, ha a szokásos KSZI helyett XI, a PHI helyett FI írásmódot alkalmazunk.



36. ábra. Független túlcsordulási terület

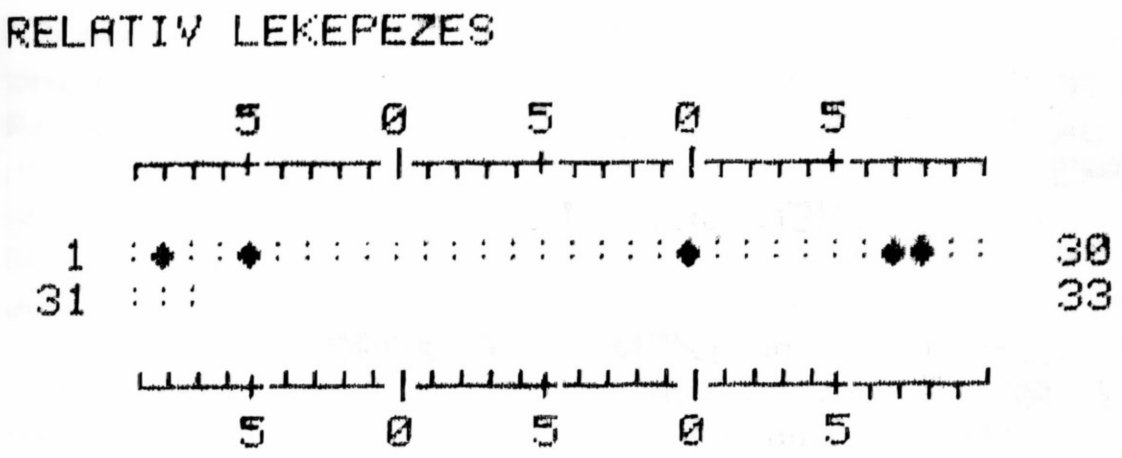
Erre nincs szükség, ha a szinonimák kezelését úgynevezett független túlcsordulási területtel oldjuk meg. Ennek az a lényege, hogy az állomány területét két részre osztjuk (36 ábra):

Az első rész az 1-től 26-os címig tart. Ide képezzük le az elsődleges adatokat. Ezért ezt a területet adatterületnek hívjuk. A második rész a 27-től a 33-as címig tart. Ide tesszük azokat a rekordokat, amelyek az adatterületre már nem férnek rá, vagyis a szinonimákat. Ezért ezt a részt túlcsordulási területnek nevezzük. (Esetünkben erre elég 7 rekordnyi hely, hiszen több szinonimánk a görög betűk között nem lehet.)

A túlcsordulási területet úgy kezeljük, hogy a rekordot leképezzük az adatterületre, ekkor megkapjuk az elsődleges címet. Ha a hely szabad, oda visszük fel a rekordot. Ha nem, akkor keresünk egy szabad helyet a túlcsordulási területen. Ennek a címe lesz a másodlagos cím, és ide visszük fel a rekordot. Például:

azonosító:	BETA	cím:	2	(adatterület)
azonosító:	EPSZILON	cím:	5	(adatterület)
azonosító:	ETA	cím:	27	(túlcsordulási terület)
azonosító:	THETA	cím:	20	(adatterület)
azonosító:	TAU	cím:	28	(túlcsordulási terület)

```
1010 R=0
1020 L=ASC(R$+"0")-64
1030 IF L<1 OR L>26 THEN GOTO 1999
1040 IF RE$(L)=":"  THEN GOTO 1210
1110 :  FOR L=27 TO 33
1120 :  :  IF RE$(L)=":" THEN GOTO 1210
1130 :  NEXT L
1199 :  GOTO 1999
1210 R=L
1999 RETURN
```



Visszaolvasáskor a rekordot az adatterületen keressük. Ehhez nincs másra szükségünk, mint az elsődleges címre. Ha az ott lévő rekord üres, nincs értelme tovább keresni. Ha viszont az elsődleges címen nem a keresett rekord van, akkor azt a túlcsordulási területen kell keresni.

Az ilyen túlcsordulási területre jellemző, hogy az adatterületen kívül helyezkedik el, és bármely rekord szinonimáját tartalmazhatja. Ezért független túlcsordulási területnek nevezzük. Használata általában akkor gazdaságos, ha a szinonimák száma a rekordok számához képest kicsi.

Megjegyezzük, hogy a példában a szinonimák mindig a túlcsordulási terület legelső szabad helyére kerülnek, a címük tehát nem jellemző a rekordra. Lehetne ez másként is; alkalmazhatnánk külön címképző eljárást, úgynevezett másodlagos leképezést a túlcsordulási területen belül.

A bemutatott eljárás persze csak a görög betűk neveire működik, egy munkahely dolgozóinak a neveire már nem. Egyrészt ott szinte minden betűnél több szinonima fordul elő, másrészt azonos nevek is lehetnek. Ilyenkor más leképezési eljárást kell választanunk. Tegyük fel, hogy az állományban minden rekordnak legfeljebb négy szinonimája lehet, azaz egy adott betűvel legfeljebb öt rekord azonosítója kezdődhet. Az egyszerűség kedvéért zárjuk ki az azonos nevek lehetőségét. (Legyenek például az azonosítók az európai fővárosok nevei.)

Ilyenkor a szinonimákat úgynevezett alárendelt túlcsordulási területen is elhelyezhetjük. Ennek a lényege az, hogy az adatterületen belül van, éspedig minden elsődleges címhez tartozik egy-egy ezt követő, de nem önálló túlcsordulási terület az elsődleges címhez tartozó rekord szinonimáinak tárolására (37. ábra).

Itt tehát a túlcsordulási területek két elsődleges cím közé ékelődnek be, és mindig csak az előttük álló elsődleges címen lévő rekord szinonimáit tartalmazhatják; ezért nem függetlenek.

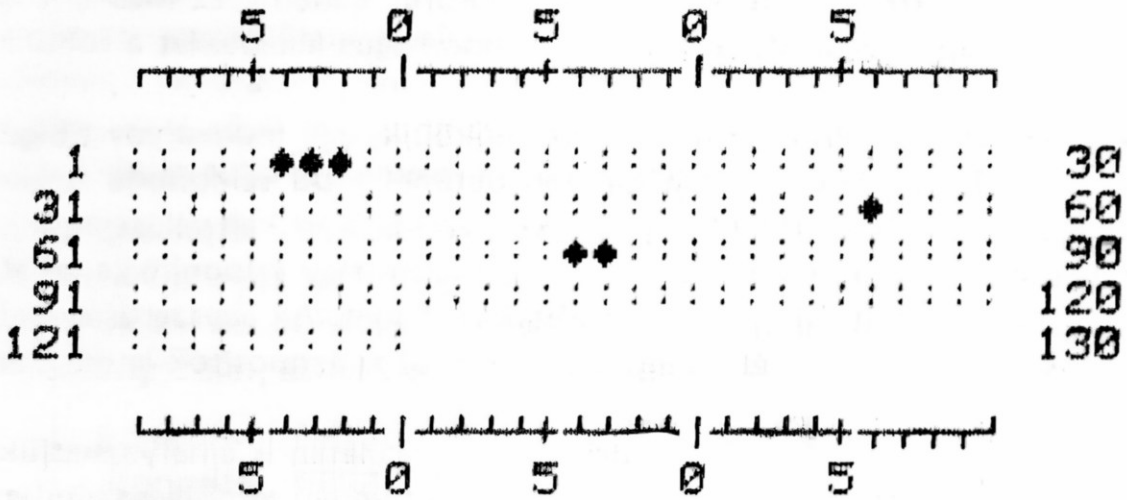
Felíráskor nyilván előbb leképezzük a rekordot az elsődleges címre, majd megnézzük, hogy ez szabad-e. Ha igen, letároljuk a rekordot, ha nem, vesszük a soron következő címet. Ha ez szabad, oda tesszük a rekordot, de ha nem az, továbblépünk mindaddig, amíg egy szabad helyet nem találunk, vagy el nem érjük a következő elsődleges címet.

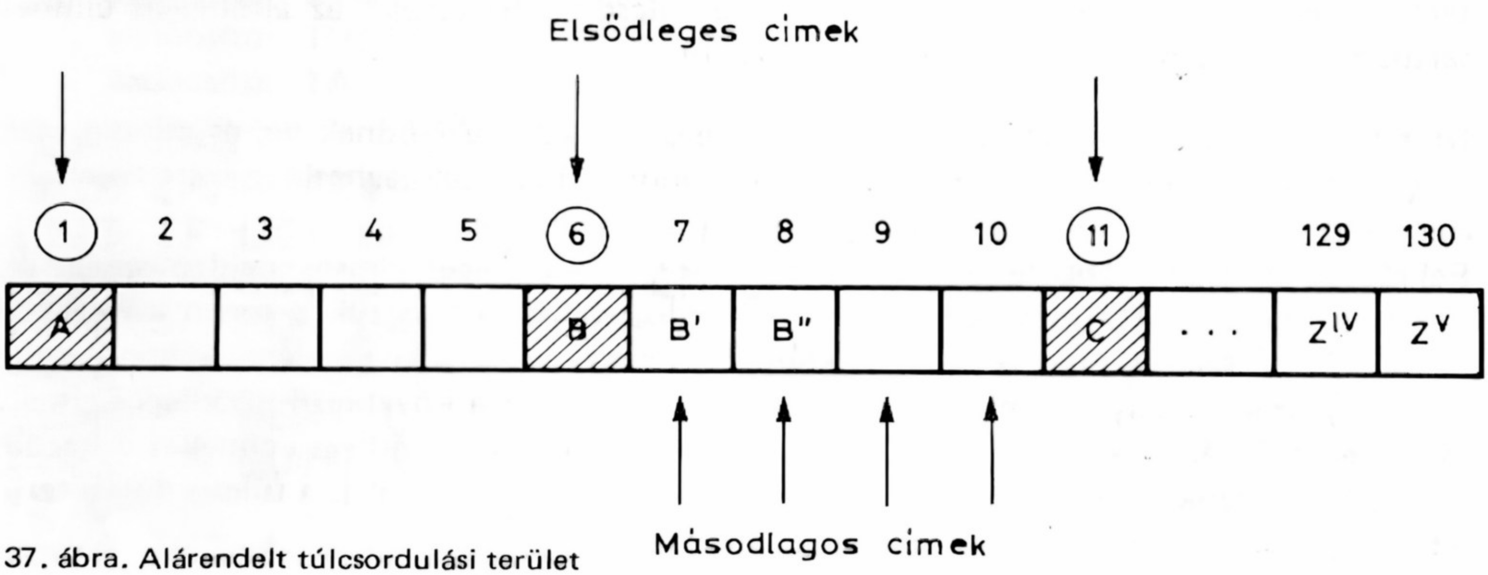
Most tehát az állományunk $26 * 5 = 130$ rekordból áll. Az elsődleges címeket a kezdőbetűből képezzük: 1, 6, 11, 16, 21 stb. Vagyis $(L-1) * 5 + 1$, ahol L a túlcsordulási terület nélküli cím:

```
1010 R=0
1020 L=ASC(R$+"0")-64
1030 IF L<1 OR L>26 THEN GOTO 1999
1040 L=(L-1)*5+1
1050 IF RE$(L)=":"  THEN GOTO 1210
1060 :  LL=L+5
1110 :  L=L+1
1120 :  :  IF L>=LL      THEN GOTO 1999
1130 :  :  IF RE$(L)=":" THEN GOTO 1210
1140 :  GOTO 1110
1210 R=L
1999 RETURN
```

azonosító:	BUDAPEST	cím:	6	(elsődleges)
azonosító:	BERLIN	cím:	7	(másodlagos)
azonosító:	BERN	cím:	8	(másodlagos)
azonosító:	LONDON	cím:	56	(elsődleges)
azonosító:	PARIZS	cím:	76	(elsődleges)
azonosító:	PRAGA	cím:	77	(másodlagos)

RELATIV LEKEPEZES





37. ábra. Alárendelt túlcsordulási terület

E módszer előnye a jól áttekinthető állományszerkezet, és a kevés fejmozgással járó, tehát gyors keresés. Hátránya, hogy ha valamelyik túlcsordulási terület megtelik, az egész állományt át kell szervezni, ami azzal is jár, hogy a feldolgozóprogramokban a leképező eljárásokat is át kell dolgozni. A független túlcsordulási területnél erre nincs szükség, az állomány automatikusan kiterjeszthető.)
Elvileg szervezhető olyan relatív állomány is, amelynek egyaránt van alárendelt és független túlcsordulási területe. Ilyenkor nemcsak másodlagos, hanem harmadlagos címekkel is számolnunk kell. (Általában a másodlagos címek az alárendelt, a harmadlagosak a független túlcsordulási területen helyezkednek el; azaz ha egy rekord nem tárolható az elsődleges címen, akkor a cím alárendelt túlcsordulási területre kerül, ha pedig az is betelt, akkor a független túlcsordulási területre tesszük.) Az ilyen állomány leképezési eljárása a legtöbbször igen bonyolult, ami már önmagában is elegendő ahhoz, hogy gyakorlati alkalmazása csak nagyon szűkkörű legyen.

Megjegyezzük, hogy a bemutatott példák messze nem merítik ki a leképezési eljárások lehetőségeit. Csupán szemelvények egy szinte kimeríthetetlen témakörből.
A tesztprogramokba beépített leképező eljárások rutinjai sem gyakorlati alkalmazás céljára, hanem a leképezés kipróbálására készültek. A gyakorlatban ugyanis a leképezést nemcsak felvitelre, hanem beolvasásra is végre kell hajtani, sőt általában minden felvitelt egy lekérdezés előz meg, amely eldönti, hogy a rekord szerepel-e már az állományban.
A tesztprogramokban e lekérdezések végrehajtását feltételeztük, és azt is, hogy a rekordot nem találta meg; tehát az felvihető.
Természetesen nincs akadálya, hogy a lekérdezésre külön tesztprogramot írjunk, vagy hogy beépítsük az említett programokba. Erre azonban nincs szükség, mert a következő fejezetben bemutatunk egy olyan programot, amelynek leképező rutinjai már gyakorlati igényeket is kielégíthetnek.

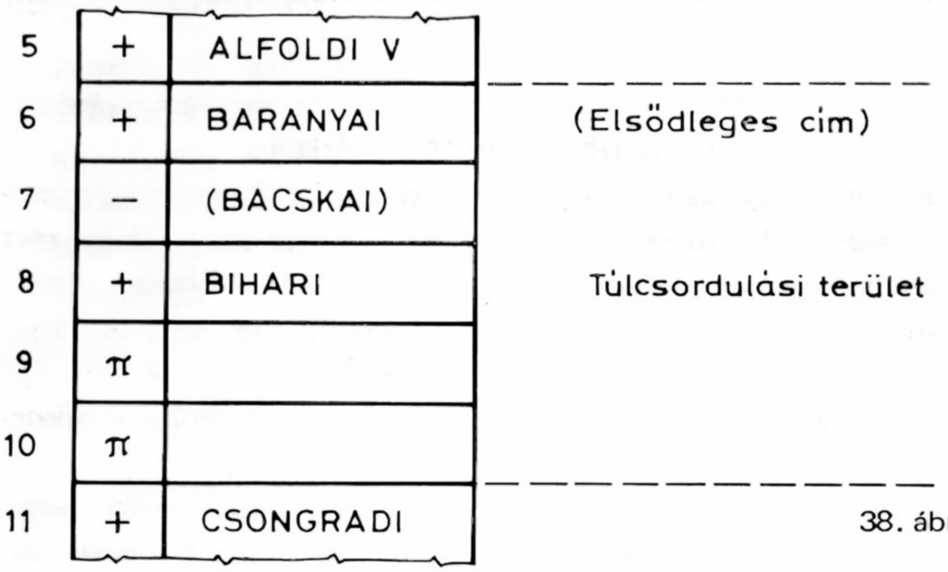
RELATÍV ÁLLOMÁNY TÚLCSORDULÁSI TERÜLETTEL

A leképezési eljárások és a relatív állományok kezelésének gyakorlására készítsünk egy házi telefonkönyvet, amely a következő adatokat tartalmazza:

- T$: állapotjelző (1 bájt)
- S$: szeparátorjel (1 bájt)
- K$: név (25 bájt)
- S$: szeparátorjel (1 bájt)
- L$: lakás telefonszáma (7 bájt)
- S$: szeparátorjel (1 bájt)
- M$: munkahely telefonszáma (12 bájt)
- S$: szeparátorjel (1 bájt)

Ha a név hosszabb 25 bájtnál, rövidíteni kell. A lakás és a munkahely telefonszáma nem foglalja magában a (távhíváshoz szükséges) körzetszámot, de tartalmazza a szokásos kötőjelet. A munkahelyi számban ezenkívül egy „/" jel és egy mellékállomás legfeljebb négyjegyű hívószáma is szerepelhet. A teljes rekord mérete tehát 49 bájt.

Ha kívánjuk, a mezőket megnövelhetjük, hogy bővebb információt (több melléket, körzetszámot stb.) tartalmazhassanak, de ennek nincs különösebben értelme, hiszen a programot végül is nem közvetlen gyakorlati felhasználásra, hanem kísérletezésre szánjuk.
Az állomány legyen olyan, mint a szokásos betűregiszteres zsebnotesz, amelyben minden betű új lapon kezdődik, és minden lapon üres rovatok vannak a bejegyzések számára. Az egyszerűség kedvéért állapodjunk meg abban, hogy ezen rovatok száma laponként öt; a teszteléshez ennyi is elég.
Az angol ábécé 26 betűből áll, ezért állományunk 26*5 = 130 rekord tárolására alkalmas. A képzeletbeli lapoknak megfelelően az 1–5 rekordokat az A betűvel kezdődő neveknek, a 6–10 rekordokat a B betűseknek, és így tovább, a 126–130 rekordokat pedig a Z-vel kezdődőeknek tartjuk fenn.



38. ábra. Állomány túlcsordulási területtel (Telefonkönyv állomány)

A 38. ábrán jól látható, hogy tulajdonképpen a már ismert alárendelt túlcsordulási területet alkalmaztuk. Azonban el kell térnünk az előző fejezetben bemutatott rekordkezelési módtól, hiszen az, amint már ott is felhívtuk rá a figyelmet, csak az új rekordok elhelyezésére alkalmas.
Hogyan kezeljük hát a rekordokat? Úgy, mint a zsebnoteszt:

- Ha egy rekordot fel akarunk vinni, akkor beírjuk az első szabad helyre, feltételezve, hogy azon a lapon, amelyen lennie kell, a rekord még nem szerepel, és van szabad hely.
- Ha egy rekordot törölni akarunk, töröljük az adatait, feltéve, hogy azon a lapon, amelyen lennie kell, a rekord már szerepel.
- Ha egy rekordot módosítani akarunk, az adatait felülírjuk, ha azon a lapon, amelyen lennie kell, a rekord már szerepel.
- Ha egy rekordot le akarunk kérdezni, akkor beolvassuk az adatait, persze csak akkor, ha azon a lapon, amelyen lennie kell, a rekord már szerepel.

Mind a négy művelet két fontos lépésre épül: a notesz felütése a megfelelő lapon, és a bejegyzés megkeresése a lapon belül.
A felvitel esetén még egy lépés van: ha a bejegyzés nincs a lapon, akkor szabad helyet kell neki keresni.

Mindezeket figyelembe véve úgy célszerű felfogni relatív állományunkat, hogy abban egyáltalán nincs adatterület, hanem kizárólag túlcsordulási területből áll.
Más szóval ez azt jelenti, hogy a túlcsordulási terület nem az elsődleges címet követő címen kezdődik, hanem magán az elsődleges címen, vagyis az első másodlagos cím azonos az elsődleges címmel. (A megfogalmazás most szőrszálhasogatásnak tűnhet, de később ki fogjuk használni ezt a tulajdonságot.)
Mi az előnye ennek a szervezési módnak? Elsősorban az, hogy a túlcsordulási terület megfelel a notesz egy teljes lapjának. Így egységes kereső eljárást írhatunk, hiszen a notesznek (állománynak) nincs kitüntetett rovata (rekordja) egy lapon (leképezési területen) belül.
A fenti koncepciónak megfelelően a leképező eljárás két részből áll: az első másodlagos cím (vagyis az elsődleges cím) meghatározásából, valamint a keresésből.
A címet a már ismert módon számíthatjuk ki, némileg persze a feladat igényeinek megfelelően, testre szabva:

```
2010 B$=LEFT$(N$,1)
2020 RL=(ASC(B$)-65)*5+1
2030 RH=0
2999 RETURN
```

Az eljárás bemenő paramétere az N$ név, vagyis a rekordazonosító. Ennek az első B$ betűje (bájtja) határozza meg az elsődleges cím RL alsó bájtját. Az RH felső bájt pedig mindig nulla, hiszen a legmagasabb cím 130 lehet, ami bőven 255 alatt van.
Látható, hogy a címkiszámítás meg sem adja a rekordsorszámot, hanem egyenesen a konvertált formát hozza létre; éspedig nem az általános képlettel, hanem a feladatnak megfelelő, egyszerűsített algoritmussal.

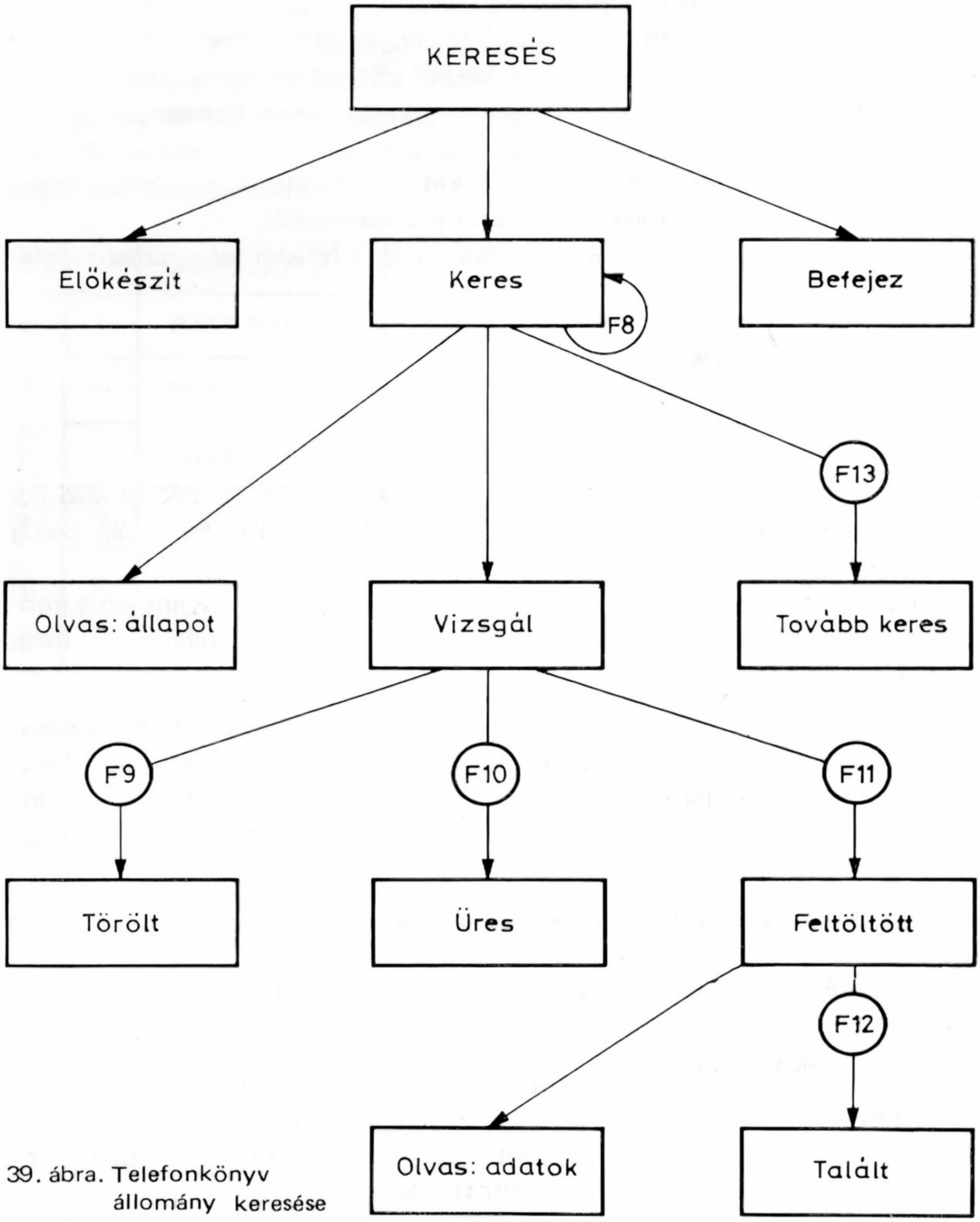
Fordítsuk most figyelmünket a kereső eljárásra. Minthogy a keresés a rekordok olvasásával jár, a program sebessége szempontjából lényeges, hogy a keresés a túlcsordulási területen belül csak egyszer olvassa végig a rekordokat. Tehát a keresésnek egyetlen menetben kell megtalálnia a keresett rekordot, vagy ha nincs benne az állományban, akkor az első szabad helyet.

Kérdés, hogy mit tegyünk a törölt rekordokkal. Őrizzük meg, vagy írjuk felül őket? A feladat jellegéből következik, hogy a megőrzésnek nincs sok értelme. Ha tehát a törölt rekordok felülírását megengedjük, akkor célszerű őket szabad helynek tekinteni. Ennek van egy határozottan előnyös következménye, amit akkor érthetünk meg igazán, ha megvizsgáljuk a szabad helyek szerepét:

- Ha a túlcsordulási terület üres, az első szabad hely az első címen van.
- Ha a túlcsordulási terület nem üres, de nincs is tele, ugyanakkor nincsenek benne törölt rekordok, az első szabad hely az utolsó feltöltött rekord utáni címen van.
- Ha a túlcsordulási területen vannak törölt rekordok, az első szabad hely az első törölt rekord címén van.
- Ha a túlcsordulási terület megtelt, és nem tartalmaz törölt rekordot sem, akkor nincs benne szabad hely.

Mindezeknek az a következménye, hogy a túlcsordulási terület a kezdőcímtől kezdve folyamatosan fog feltöltődni; feltöltetlen rekordok csak a feltöltött rész után fordulhat-

nak elő. A feltöltés folyamatosságát persze a törölt rekordok időlegesen megszakíthatják ugyan, de üres rekordok soha. Ez pedig azért előnyös, mert a keresést nem szükséges a teljes túlcsordulási területre végrehajtani, elég az első feltöltetlen rekordig; ha létezik ilyen. Nyilván végig kell keresnünk a területet, ha megtelt.
Ennek megfelelően a keresési eljárást a 39. ábra mutatja.



39. ábra. Telefonkönyv állomány keresése

Ahol a feltételek:

– F8: ismétlődik az RL elsődleges címtől 1-esével legfeljebb 5 rekordra, amíg üres rekordot nem talál, vagy amíg meg nem találja a keresett rekordot, azaz mindaddig ismétlődik, amíg az I<=5 és az F13 feltétel teljesül;

– F9: ha a rekord törölt, azaz a T$ állapotjelző ="–";
– F10: ha a rekord feltöltetlen, tehát a T$ állapotjelző =π;
– F11: ha a rekord feltöltött, vagyis a T$ állapotjelző ="+";
– F12: ha a rekordból beolvasott K$ azonosító megegyezik az adott N$ névvel;
– F13: ha nem talált, de még van nem üres rekord, azaz ha nem F12 és nem F10.

A rutin bemenő paraméterként a keresendő N$ nevet, valamint az ennek megfelelő alárendelt túlcsordulási terület kezdőcímét, az RL elsődleges címet kéri.

Kimenő paramétere az RL rekordcím, valamint két jelző, a TALALT és a MEGTELT. Az utóbbiakat logikai változóként kezeli, és a nevüknek megfelelően „igaz" vagy „hamis" értékre állítja be őket.

Ha TALALT rekordot, akkor az RL ennek a címét tartalmazza, ha nem, akkor az első szabad helyét. Ha van törölt rekord a túlcsordulási területen, akkor az RL az első törölt rekord címe, ha nincs, akkor az első feltöltetlen (üres) rekordé. Ha a túlcsordulási terület MEGTELT, azaz ha nincs benne sem törölt, sem üres rekord, akkor az RL nem tartalmaz rekordcímet, vagyis az értéke nulla.

Megjegyezzük, hogy a TALALT és MEGTELT logikai változók pusztán a programozó kényelmét szolgálják. A rekord állapotát és helyzetét az RL címmel, valamint a T$ állapotjelzővel a fenti logikai jelzők nélkül is egyértelműen meghatározhatnánk. A TALALT és MEGTELT jelzők logikai értékei, amelyeknek a tagadására (NOT) is rá lehet kérdezni, megkönnyítik a keresés eredményének kiértékelését.

A rutin kódolva így néz ki:

```
3010 MEGTELT=0
3020 TALALT =0
3030 SZ     =0
```

Azzal kezdődik, hogy nullára állítja a MEGTELT és a TALALT jelzőt, valamint az első szabad hely SZ címét.

```
3110 FOR I=1 TO 5
```

Ezután szervez egy ciklust, amely végigmegy a túlcsordulási terület öt rekordján, az RL kezdőcímtől kezdve.

```
3120 :   RC=1:GOSUB 8010
```

A cikluson belül először beolvassa az aktuális címen lévő rekord T$ állapotjelzőjét az RC=1 bájtpozícióról.

Ezután megvizsgálja a rekord állapotát, vagyis hogy a rekord törölt, üres vagy feltöltött-e. Ha egyik sem (ami elvileg lehetetlen, de hiba esetén előfordulhat), tovább keres:

```
3130 :   IF T$="-" THEN GOTO 3210
3140 :   IF T$="π" THEN GOTO 3310
3150 :   IF T$="+" THEN GOTO 3410
3199 :   GOTO 3510
```

Ha a rekord törölt volt, megjegyzi az RL címét, mert ezt tekinti a továbbiakban az első SZ szabad helynek, persze csak akkor, ha még nem volt szabad hely, azaz ha SZ=0. Majd tovább keres:

```
3210 :   :  IF SZ=0 THEN SZ=RL
3299 :   GOTO 3510
```

Ha a rekord üres, akkor megjegyzi az RL címét, és azt tekinti az első SZ szabad helynek, de csak akkor, ha eddig még nem talált szabad helyet. Ezután abbahagyja az immár reménytelen keresést, hiszen üres rekord csak a túlcsordulási terület feltöltött része után következhet:

```
3310 :  :  IF SZ=0 THEN SZ=RL
3399 GOTO 3910
```

Ha a rekord feltöltött, akkor beolvassa az adatait az RC=2 bájttól kezdve. Majd megvizsgálja, hogy a rekord azonos-e a keresettel, azaz a megadott N$ név megegyezik-e a beolvasott K$ névvel. Ha nem, tovább keres, viszont ha megtalálta a keresett rekordot, akkor az RL címet tekinti az első SZ szabad helynek, és egyben TALALT jelzést ad, majd természetesen abbahagyja a keresést:

```
3410 :  :  RC=2:GOSUB 8010
3420 :  IF N$<>K$ THEN GOTO 3510
3430 :  :  SZ=RL:TALALT=(SZ<>0)
3499 GOTO 3910
```

A továbbkeresés abból áll, hogy veszi a soron következő RL címet, és visszatér a kereső ciklus elejére:

```
3510 :  RL=RL+1
3599 NEXT I
```

A keresés befejezésekor beállítja a kimenő paramétereket. Az RL rekordcímet az első SZ szabad hely címére (ami jelenthet foglalt helyet is, ha a rekordot megtalálta); a MEGTELT jelzőt pedig „hamis" vagy „igaz" (0 vagy —1) értékre, attól függően, hogy talált-e rekordot, illetve szabad helyet, avagy sem:

```
3910 RL=SZ
3920 MEGTELT=(SZ=0)
3999 RETURN
```

A TALALT jelzőt nem kell külön állítania, hiszen az értéke induláskor eleve „hamis" volt. Ezt azonnal átállította „igaz"-ra, amint a rekordot megtalálta. Ha nem találta meg, akkor maradnia kell a „hamis" értéknek.
Látható, hogy a terület feltöltésekor az új rekord elsősorban törölt rekord helyére kerül; üres rekordot a keresés csak akkor választ, ha a törölt rekordok elfogytak. Ennek az az előnye, hogy a keresést mindig a lehető legkevesebb rekordon kell végrehajtani, ami a feldolgozást gyorsítja.
Mindenesetre írjuk meg a tesztprogramot a leképezés kipróbálására. A "TELEFONTORZS" állományt a 130 elemű TT$ tömbbel szimuláljuk. A tömbelemek csak az állapotjelzőt és a rekordazonosítót fogják tartalmazni, szeparátorjelek nélkül; hiszen csak a keresésre épülő leképező eljárást, nem pedig magát a karbantartást akarjuk tesztelni. Induláskor a TT$ tömb, mint a frissen létrehozott relatív állomány, π jelekkel van feltöltve:

```
110 DIM TT$(130)
210 FOR I=1 TO 130
220 :  TT$(I)="π"
230 NEXT I
```

Ezután következik a tulajdonképpeni tesztelés. A program bekéri a J$ állapotjelzőt, és ha ez nem "*" leállítójel, akkor az N$ rekordazonosítót is, majd leképezi a rekordot. Ezt mindaddig ismétli, amíg le nem állítjuk:

```
310 PRINT"▒▒ TELEFONTESZT▒▒"
320 :   INPUT"▒ JEL = ";J$
330 :   IF J$="*" THEN GOTO 910
340 :   IF J$<>"+" AND J$<>"-" THEN GOTO 310
350 :   :  INPUT"▒ NEV = ";N$
360 :   :  GOSUB 2010
370 :   IF RL<1 OR RL>126 THEN GOTO 310
380 :   :  GOSUB 3010
390 :   :  GOSUB 6010
399 GOTO 310
```

Állapotjelzőnek J$ = "+" jelet kell megadnunk, ha a rekord felvitelét vagy módosítását akarjuk szimulálni; ha pedig a törlést szeretnénk kipróbálni, J$ = "–" jelet kell begépelnünk. A programot a J$ = "*" jellel állíthatjuk le. Más J$ jelet a program nem fogad el. Az állapotjelző után adhatjuk meg az N$ rekordazonosítót, azaz a nevet. Ennek feltétlenül betűvel kell kezdődnie, amit a program ellenőriz.

A rekord leképezése két lépésben történik: először előállítja az elsődleges címet (2010), majd végrehajtja a keresést (3010).

A keresés eredményétől függően hajtja végre a rekord felírását az állományba (6010), amit itt a megfelelő tömbelem feltöltésével szimulálunk.

Ezzel a leképezés szimulálása tulajdonképpen készen van. Az eredmény ellenőrizhetősége érdekében a program alkalmas a TT$ tömb tetszőleges A alsó és F felső határ közötti intervallumokban való kiírására:

```
910 PRINT"▒▒ TELEFONTORZS"
920 :   INPUT"▒ REKORDTOL =";A
925 IF A=0 THEN GOTO 999
930 :   INPUT"▒ REKORDIG  =";F
940 IF A<1 OR F>130 OR A>F THEN GOTO 910
950 :   PRINT"▒"
960 :   FOR I=A TO F
961 :   :  PRINT RIGHT$("    "+STR$(I),4);
962 :   :  PRINT" ";LEFT$(TT$(I),1);
963 :   :  IF LEFT$(TT$(I),1)="π" THEN GOTO 968
964 :   :  PRINT" ";MID$(TT$(I),2,25);
968 :   :  PRINT
969 :   NEXT I
970 GET X$:IF X$="" THEN GOTO 970
980 GOTO 910
999 END
```

Akárhány intervallum kiíratható, bármilyen sorrendben. Az intervallum A és F határait tetszés szerrint választhatjuk meg, de a program ellenőrzi, hogy csak a címtartományon belül adhassunk meg értékeket. A kiírás A=0 paraméterrel állítható le.

Mivel a TT$ tömbelemek nem tartalmaznak szeparátorjelet, az olvasható kiíráshoz két részre kell bontani őket: az első bájt adja az állapotjelzőt, a második bájttól kezdve (legfeljebb 25 karakter hosszúságban) helyezkedik el a név. A program ezeken kívül minden rekordhoz kiírja az I rekordcímet (indexet) is.

Az intervallum kiírása után programunk várakozik, hogy legyen időnk a képernyő tanulmányozására, Bármilyen gomb megnyomására továbblép, és kéri a következő intervallumot.

Most már csak a szubrutinok vannak hátra. Először nézzük a korábban már bemutatott címkiszámító eljárást:

```
2010 B$=LEFT$(N$,1)
2020 RL=(ASC(B$)-65)*5+1
2030 RH=0
2999 RETURN
```

Majd az imént részletesen ismertetett kereső eljárást:

```
3010 MEGTELT=0
3020 TALALT =0
3030 SZ     =0
3110 FOR I=1 TO 5
3120 :   RC=1:GOSUB 8010
3130 :   IF T$="-" THEN GOTO 3210
3140 :   IF T$="π" THEN GOTO 3310
3150 :   IF T$="+" THEN GOTO 3410
3199 :   GOTO 3510
3210 : :   IF SZ=0 THEN SZ=RL
3299 :   GOTO 3510
3310 : :   IF SZ=0 THEN SZ=RL
3399 GOTO 3910
3410 : :   RC=2:GOSUB 8010
3420 :   IF N$<>K$ THEN GOTO 3510
3430 : :   SZ=RL:TALALT=(SZ<>0)
3499 GOTO 3910
3510 :   RL=RL+1
3599 NEXT I
3910 RL=SZ
3920 MEGTELT=(SZ=0)
3999 RETURN
```

A korábbi tesztprogramjainkkal ellentétben itt a leképezés közvetlen összefüggésben áll az állomány kezelésével, ezért szükségünk van még két modulra: az állományból rekordot olvasó, valamint az állományba rekordot felíró szubrutinra.

Ezek közül az olvasás az egyszerűbb. Természetesen nem igazán olvasunk, hanem csak szimuláljuk. Ha a bájtpozíció RC = 1, akkor csak a T$ állapotjelzőt, ha pedig a bájtpozíció RC = 2, akkor csak a K$ nevet olvassuk az RL címen tárolt rekordból, azaz vesszük ki a TT$ tömb RL indexű eleméből:

```
8010 IF RC=1 THEN T$=LEFT$(TT$(RL),1)
8020 IF RC=2 THEN K$=MID$(TT$(RL),2,25)
8999 RETURN
```

A rekord felírását is nyilvánvalóan csak szimuláljuk: a J$ állapotjelzőt és az N$ nevet a TT$ tömbnek a kereső eljárás által megadott RL-lel indexelt elemébe töltjük be:

```
4010 IF TALALT  THEN GOTO 4210
4020 IF MEGTELT THEN GOTO 4310
4110 PRINT"▓▓ NEM TALALT"
4120 PRINT "▓ SZABAD HELY = ";RL
4199 GOTO 4510
4210 PRINT"▓▓ TALALT"
4220 PRINT "▓ CIME      = ";RL
4230 PRINT "▓ TARTALMA = ";TT$(RL)
4299 GOTO 4510
4310 PRINT "▓▓ MEGTELT"
4510 GET X$: IF X$="" THEN GOTO 4510
4610 IF MEGTELT THEN GOTO 4999
4620 IF J$="-" THEN N$="O"
4630 TT$(RL)=J$+N$
4999 RETURN
```

Ezt persze csak akkor tehetjük meg, ha a keresés a rekordot megtalálta, vagy ha nem, akkor talált (feltöltetlen vagy törölt) szabad helyet. Ezért figyelnünk kell a kereső eljárás kimenő paramétereit, a TALALT és a MEGTELT jelzőket. A program tájékoztatás végett ezeknek megfelelő üzenetet ad a képernyőre, feltüntetve a címet, amelyre az eljárás a rekordot leképezte, sőt megtalált rekord esetén annak tartalmát is.

Az üzenetek olvasására is hagy időt a program, csak gombnyomásra (bármely gomb megnyomására) megy tovább, és kéri a következő rekord állapotjelzőjét, továbbá azonosítóját.

Ha begépeltük a programot, mentsük ki TELEFONTESZT néven, majd futtassuk le. Vigyünk fel tetszés szerinti rekordokat, bár célszerűbb a korábban bemutatott mintaállományt létrehozni. Például:

```
TELEFONTESZT


JEL = ? +
NEV = ? BACSKAI

NEM TALALT
SZABAD HELY =  7
```

FIGYELEM!
Az üzenetek megjelenése után a program várakozik. Csak akkor lép tovább, ha valamilyen gombot megnyomunk.

Próbáljunk meg létező rekordot törölni. Például:

```
TELEFONTESZT


JEL = ? -
NEV = ? BACSKAI

TALALT
CIME      =  7
TARTALMA = +BACSKAI
```

Vigyünk fel azonos betűvel kezdődő neveket is, mégpedig annyit, hogy a túlcsordulási terület megteljen. Például:

```
TELEFONTESZT


JEL = ? +
NEV = ? ALFOLDI VI

MEGTELT
```

Ha elég rekordot vittünk fel, módosítottunk, illetve töröltünk, adjuk J$="*" állapotjelzőt. Ekkor a program áttér a tömb kiírására, és bekéri az intervallumot. Ha megkapta, kiírja a tömb megfelelő részét:

```
TELEFONTORZS
REKORDTOL =? 5
REKORDIG  =? 11

  5 + ALFOLDI V
  6 + BARANYAI
  7 - 0
  8 + BIHARI
  9 π
 10 π
 11 + CSONGRADI
```

Az intervallum végén várakozik, míg egy tetszőleges gombot meg nem nyomunk. Akkor új intervallumot kér. Így akár a teljes tömböt is kiírathatjuk. Ezt célszerű több lépésben, legfeljebb 15 rekordonként megtenni, mert a hosszabb intervallumok nem férnek ki a képernyőre.
Ha már nem vagyunk kíváncsiak több intervallumra, adjunk REKORDTOL=0 alsó határt. Ekkor a program leáll.

Sokan úgy vélik, hogy felesleges ilyen tesztprogramokat írnunk, hiszen az „éles" programot amúgy is tesztelnünk kell.

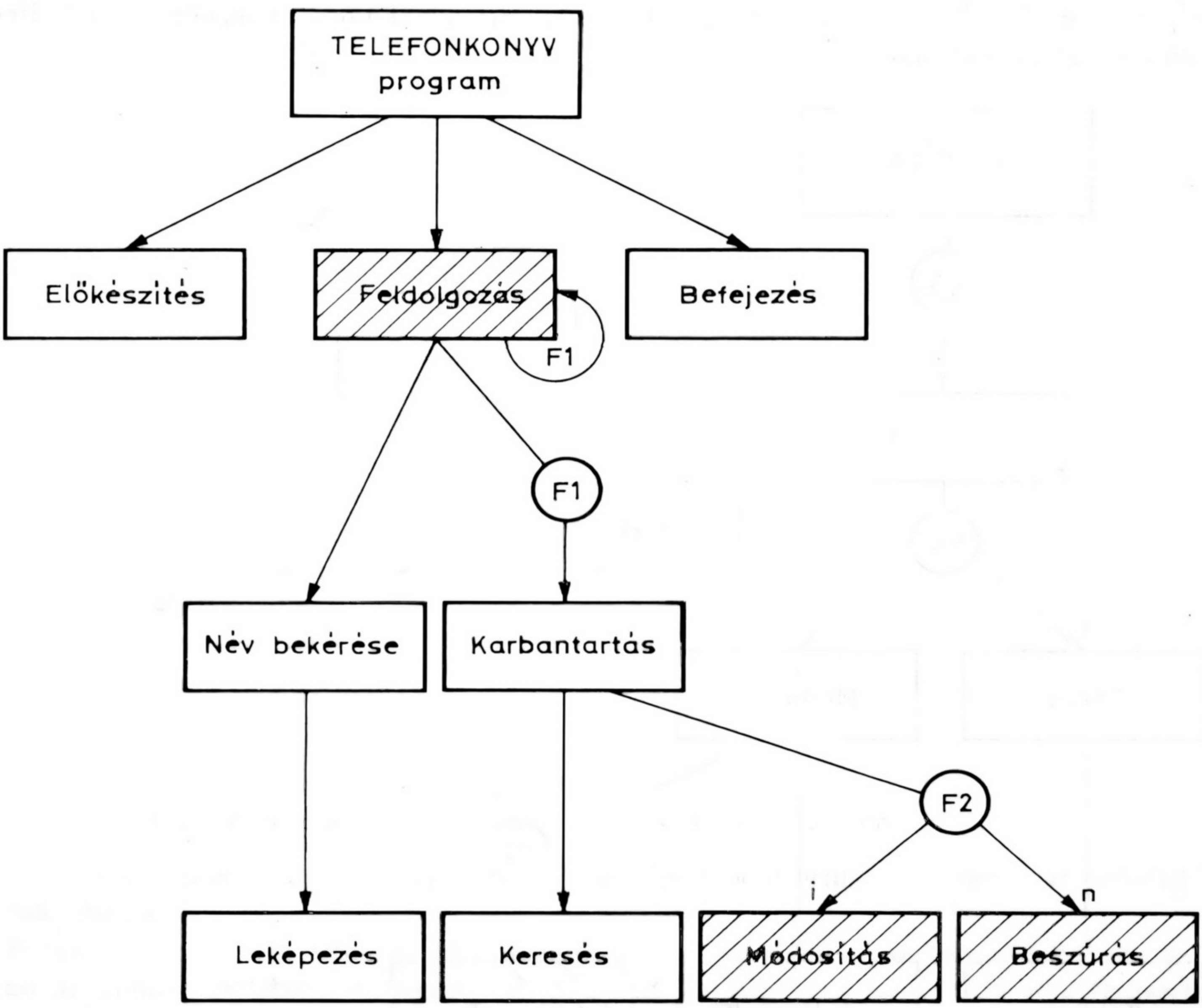
Elsősorban a kezdő programozók esnek (önhibájukon kívül) abba a pszichológiai csapdába, hogy nemcsak egyszerűen többletmunkának érzik a szimulált tesztelést, hanem a teljesítési (átadási) határidőt veszélyeztető időfecsérlésnek is tartják.

Ezzel szemben ezek a tesztprogramok eléggé sablonosak: mielőtt meghívnák a tesztelendő eljárást, bekérik a bemenő paramétereit, az eljárás végrehajtása után pedig kiírják az általa előállított kimenő paramétereket. A tesztelendő eljárásból hívott szubrutinok helyett általában csak az eredeti rutin hatását szimuláló modulokat tartalmazzák.

Ha van már néhány mintánk, egy ilyen tesztelő programocskát egy órán belül meg lehet írni. Ezek után a tesztelés már valóban csak percek kérdése, feltéve, hogy a tesztadatokat jó előre gondosan megterveztük.

FIGYELEM!

Tilos találgatással tesztelni, mert ez veszélyesebb, mintha egyáltalán nem tesztelnénk. Rosszabb hogyha egy nem kielégítő tesztelés után hiszünk egy tesztelt eljárás látszólagos helyességében, mint ha tudjuk, hogy az eljárás teszteletlen, s így az eredményében nem bízunk meg.



40. ábra. Telefonkönyv I.

A bonyolult algoritmusokat mindig célszerű külön tesztelni. Amikor ugyanis a bizonyítottan helyes eljárást beépítjük egy ,,éles" programba, és az nem működik jól, bízhatunk abban, hogy a hibát nem a beépített eljárásban kell keresnünk. Ez különösen érvényes olyan eljárásokra, amelyek elég általánosak ahhoz, hogy több különböző programban is felhasználhatók legyenek. A leképező eljárások nagyrészt ilyenek.

A részeljárások külön tesztelése tehát a program tesztelését nem teszi feleslegessé, de jelentős mértékben megkönnyíti. Végső soron magát a tesztelést is meggyorsítja, és megbízhatóbbá teszi.

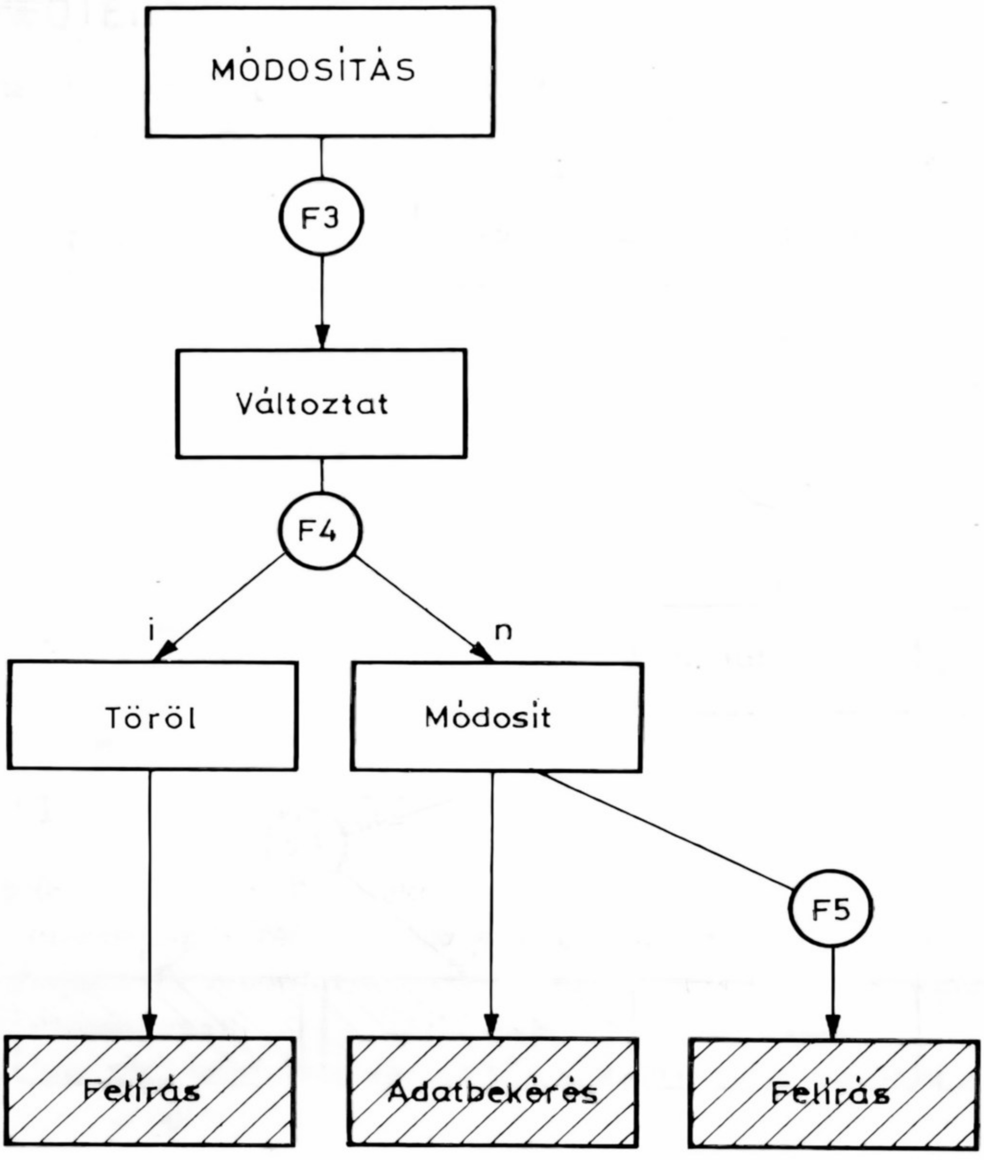
Most pedig lássuk végre a házi telefonkönyvünket kezelő programot!

A feldolgozás alapja a karbantartás, amelyhez feltétlenül szükségünk van a leképező eljárásra. Ez, mint láttuk, két részből áll: az elsődleges cím meghatározásából (ez a tulajdonképpeni leképezés), valamint a keresésből (40. ábra).

A feltételek:

- F1: ha van keresendő név, vagyis N$ nem ="*";
- F2: ha a keresés talált rekordot, azaz TALALT nem = 0.

A karbantartás meglévő rekordok módosítását vagy új rekordok felvitelét (beszúrását) engedi meg. A módosítás vagy a rekord törlését, vagy az adatainak megváltoztatását (felülírását) jelenti (41. ábra).

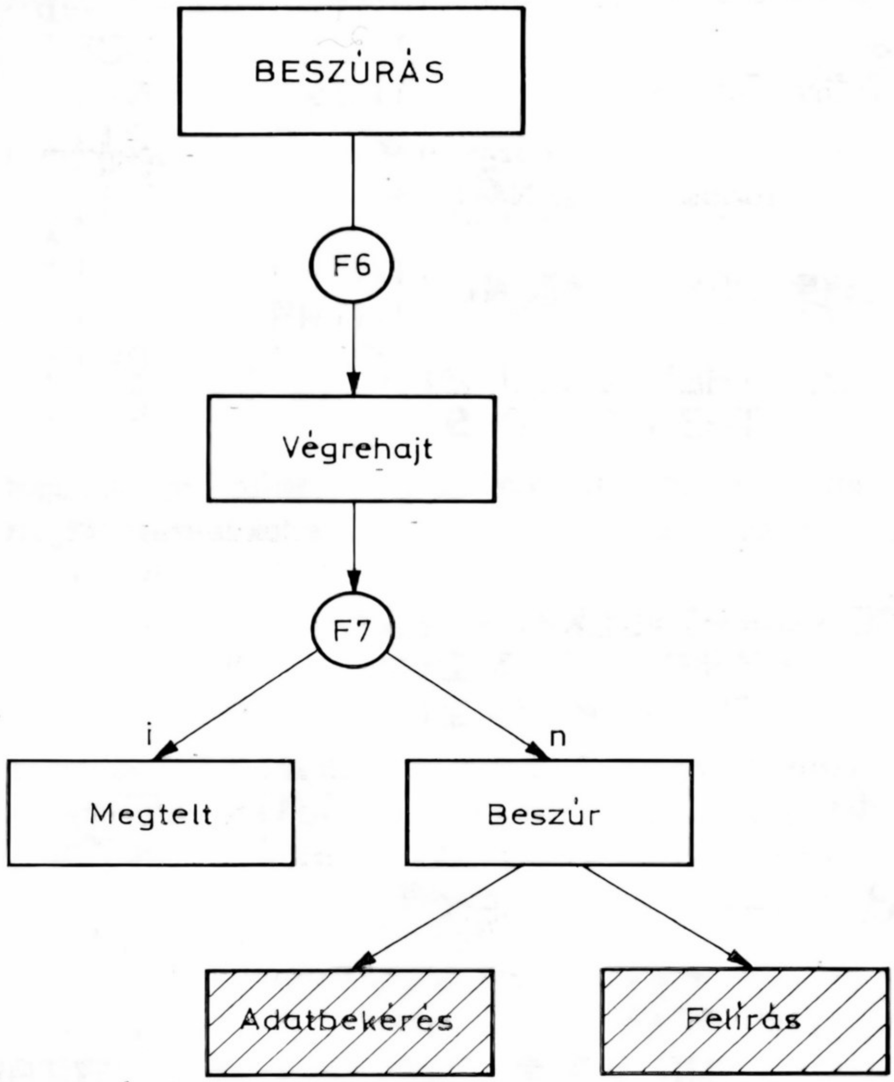


41. ábra. Telefonkönyv II.
– MÓDOSÍTÁS

Ahol a feltételek:

– F3: ha van változás, tehát a V$ válasz = "I";
– F4: ha a rekord törlendő, azaz a V$ válasz = "I";
– F5: ha van módosító adat, vagyis az LL$ lakástelefon nem = "*" vagy az MM$ munkahelyi telefon nem = "*".

A beszúrás természetesen csak akkor hajtandó végre, ha az elsődleges címnek alárendelt túlcsordulási terület még nem telt be (42. ábra).



42. ábra. Telefonkönyv III. – BESZÚRÁS

A feltételek:

– F6: ha a rekord felvihető, azaz a V$ válasz = "I";
– F7: ha a túlcsordulási terület megtelt, vagyis MEGTELT nem = 0.

Itt emlékezzünk arra, hogy az állományban nincs adatterület, minden adat azonnal a túlcsordulási területre kerül.
A programban a képernyő kezelését, valamint a program és a felhasználó közötti kapcsolat tartását a lehető legegyszerűbben oldottuk meg, hogy ne vonjuk el a figyelmet a relatív állományt kezelő funkciókról.

Ennek az elvnek megfelelően a program egy egyszerű üzenettel jelentkezik be:

```
110 S$=CHR$(13)
120 PRINT "□"
130 FOR I=1 TO 20
140 :  PRINT TAB(10);"TELEFONKONYV"
150 NEXT I
160 TH=100: GOSUB 9010 :REM: --> VARAKOZTATAS
```

Hogy a bejelentkező felirat ne tűnjön el egy szempillantás alatt, a program TH ideig várakozik. Majd megnyitja a parancscsatornát és a "TELEFONTORZS" relatív állományt:

```
170 OPEN 15,8,15
180 OPEN 2,8,2,"TELEFONTORZS"
```

Ezzel az előkészítés befejeződött. Következik a feldolgozás, melynek első lépéseként a program bekéri a karbantartandó rekord azonosítóját, az N$ nevet:

```
210 PRINT "□"
220 :  PRINT:PRINT "▒*■ NEV=  *■■■■";
230 :  INPUT N$
240 :  :  IF LEN(N$)>25 THEN GOTO 210
250 :  :  IF N$="*"     THEN GOTO 910
```

A 25 karakternél hosszabb nevet nem fogadja el. Ha a név helyett leállítójelet, csillagot kap, a program befejeződik. Ha nem, akkor az N$ névre végrehajtja a leképezést, vagyis az RL elsődleges cím meghatározását:

```
260 :  GOSUB 2010 :REM: --> LEKEPEZES
270 :  :  IF RL<1       THEN GOTO 210
280 :  :  IF RL>126     THEN GOTO 210
```

Ha a leképezett RL cím kimutat a címtartományból, új N$ rekordazonosítót kér. (Ez például akkor következhet be, ha a begépelt név nem betűvel kezdődik.) Egyébként elkezdi a karbantartást. Ehhez előbb megkeresi a karbantartandó rekordot:

```
310 :  GOSUB 3010 :REM: --> KERESES
```

Ha megtalálta, akkor a módosító, ha nem, akkor pedig a beszúró modult hívja meg. Bármelyiket is hajtotta végre, utána új rekordot kér:

```
320 :  IF     TALALT THEN GOSUB 5010:REM:--> MODOSITAS
330 :  IF NOT TALALT THEN GOSUB 4010:REM:--> UJ FELVITEL
399 GOTO 210
```

Ha a programot NEV = "*" válasszal leállítottuk, akkor befejezi a karbantartást; lezárja az állományt és a parancscsatornát, majd a bejelentkezéshez hasonló, egyszerű üzenettel elbúcsúzik és leáll:

```
910 CLOSE 2
920 CLOSE 15
930 PRINT "□"
940 FOR I=1 TO 20
950 :  PRINT TAB(10);"VISZONTLATASRA!"
960 NEXT I
999 END
```

Eddig tartott a vezérlőmodul. Most pedig lássuk a szubrutinokat.

Az állomány kezelésével kapcsolatban többször is hívjuk a hibavizsgálatot. Ez a parancscsatorna H hibakódját figyeli a már ismert módon. Ha hibát észlel, kiírja az RL rekordcímet, a H$ gépi hibaüzenetet, majd lezárja az állományt a parancscsatornával együtt, és leállítja a programot. Erről az eseményről külön U$ üzenetet ad:

```
1010 INPUT#15,H,H$,T,B
1020 IF H<20 THEN GOTO 1999
1030 :  PRINT
1040 :  PRINT " REKORD: ";RL
1050 :  PRINT
1060 :  PRINT " HIBA   :   ";"▒ ";H$;" ■"
1110 :  CLOSE 2
1120 :  CLOSE 15
1130 :  U$="LEALLITVA"
1140 :  PRINT
1150 :  PRINT TAB(10);"▒ *** ";U$;" *** ■"
1199 :  STOP
1999 RETURN
```

A leképezés két modulját, a címkiszámítást (2010—2999), valamint a keresést (3010—3999) már ismerjük. Ide ugyanabban a formában kell begépelnünk őket, ahogyan a TELEFONTESZT tesztelőprogramban szerepeltek.

A beszúró szubrutin három részből áll. Először üzenetet ad arról, hogy a rekordot nem találta, majd megkérdezi, hogy beszúrhatja-e. Ha a V$ válasz tagadó, kilép a beszúró szubrutinból. Ha igenlő, megkíséreli a beszúrást. (Egyéb V$ válasz esetén újra kérdez.)

```
4010 PRINT:PRINT:PRINT
4020 PRINT "▒─■ NINCS NYILVANTARTVA!"
4030 PRINT
4040 PRINT "▒*■ FELVIHETO? =I/N=  *";
4050 INPUT "■■■■";V$
4060 :   IF V$= "N" THEN GOTO 4999
4070 :   IF V$<>"I" THEN GOTO 4040
4080 IF MEGTELT THEN GOTO 4410
4100 REM:
```

Ha a keresés talált a rekord számára szabad helyet, beállítja a T$ állapotjelzőt feltöltött jelzésre, azaz "+" értékre; a rekord L$ és M$ adatait pedig egy adott alapértékre, a pikk jelre, majd bekéri a lakás L$ és a munkahely M$ telefonszámát. Ezeknek az adatoknak a bekérése közvetett módon történik, ugyanazzal a rutinnal, amely a módosító adatokat veszi be. Vagyis az új adatokat LL$ és MM$ módosító adatként kell megadnunk, amelyekkel a program az üres rekordot módosítja. Így állítja elő az új rekordot, amit aztán felvisz az állományba, az RL címre. Egyben ilyen értelmű U$ üzenetet ad (lista a 158. oldalon).

Ha a túlcsordulási terület megtelt, a beszúrás nem hajtható végre. Ilyenkor ennek megfelelő U$ üzenetet ad, amely arról is tájékoztat, hogy melyik B$ betű területe telt meg:

```
4110 :  T$="+"
4120 :  L$="♠"
4130 :  M$="♠"
4140 :  GOSUB 7010 :REM==> ADATBEKERES
4210 :  U$="FELVIVE"
4220 :  GOSUB 6010 :REM==> FELIRAS
4299 GOTO 4999

4410 :  PRINT:PRINT:PRINT
4420 :  PRINT TAB(17);"▩   ■"
4430 :  PRINT "  A TELEFONKONYV ▩ ";B$;
4440 :  PRINT " ■ BETUS OLDALA"
4450 :  PRINT TAB(17);"▩   ■"
4460 :  PRINT
4470 :  U$="MEGTELT"
4480 :  PRINT TAB(10);"▩ *** ";U$;" *** ■"
4490 :  TH=200: GOSUB 9010 :REM: --> VARAKOZTATAS
4999 RETURN
```

Az üzenet olvashatósága érdekében a programot itt is TH ideig várakoztatni kell.
A módosítás, tehát a megtalált rekord kezelése is három részből áll. Először kiírja a megtalált rekord tartalmát, a lakás L$ és a munkahely M$ telefonszámát. (A K$ nevet azért nem, mert a vele azonos N$ név még a képernyőn van.) Majd megkérdezi, hogy van-e a rekord tartalmában változás, és ha igen, ez törlés-e vagy módosítás:

```
5010 PRINT
5020 PRINT "▩-■ LAKAS: ";L$
5030 PRINT
5040 PRINT "▩-■ MUNKA: ";M$
5110 PRINT:PRINT:PRINT
5120 PRINT "▩*■ VALTOZAS VAN? =I/N=  *";
5130 INPUT "■■■■";V$
5140 :   IF V$="N"  THEN GOTO 5999
5150 :   IF V$<>"I" THEN GOTO 5120
5210 PRINT
5220 PRINT "▩*■ TORLENDO?       =I/N=  *";
5230 INPUT "■■■■";V$
5240 :   IF V$="N"  THEN GOTO 5310
5250 :   IF V$<>"I" THEN GOTO 5220
```

A V$ válasz bármelyik kérdésre csak "I", azaz igen, vagy "N", azaz nem lehet. Minden más válasz esetén újra kérdez.
Ha a "VALTOZAS VAN?" kérdésre tagadó V$ választ adunk, nem következik be módosítás. Ilyenkor tulajdonképpen egy lekérdezést hajtottunk végre.
Igenlő V$ válasz esetén a "TORLENDO?" kérdésre adott újabb V$ válasz dönti el, hogy törlés vagy módosítás következzék-e. Ha a törlés mellett döntöttünk, átállítja a T$ álla-

potjelzőt a "–" törlőjelre, a rekord többi adatát pedig egy egyezményes értékre, a köröcskére; majd felírja a rekordot az állományba, egyidejűleg az elvégzett műveletről az U$ üzenettel tájékoztatja a gépkezelőt:

```
5260 :   T$="-"
5261 :   N$="O"
5262 :   L$="O"
5263 :   M$="O"
5270 :   U$="TOROLVE"
5280 :   GOSUB 6010 :REM: --> FELIRAS
5290 GOTO 5999
```

Ha a módosítást választottuk, bekéri az LL$ és MM$ módosító adatokat. Ha valamelyikük értéke "*", akkor ez azt jelenti, hogy a megfelelő L$ vagy M$ eredeti adat változatlanul marad. Ha nem, akkor a rekordba a megfelelő LL$ vagy MM$ új értéket kell felvinni. A program az így módosított rekordot felírja az állományba, az RL címre. Ezután megfelelő U$ üzenetet ad:

```
5310 :   GOSUB 7010 :REM: --> ADATBEKERES
5320 IF LL$="*" AND MM$="*" THEN GOTO 5999
5330 :   U$="MODOSITVA"
5340 :   GOSUB 6010 :REM: --> FELIRAS
5999 RETURN
```

A módosítást nem hajtja végre, vagyis a rekordot nem írja vissza az állományba, ha mindkét módosító adat, az LL$ és az MM$ is csillagot tartalmaz. A rekordot felvivő szubrutin az RL és RH rekordcímre, ennek 1. bájtjára pozicionál, majd felírja a T$ állapotjelzőt, az N$ nevet, az L$ lakástelefont és az M$ munkahelyi telefont, egymástól az S$ szeparátorjellel elválasztva:

```
6010 PRINT#15,"P"CHR$(2)CHR$(RL)CHR$(RH)CHR$(1)
6020 GOSUB 1010 :REM: --> HIBAVIZSGALAT
6030 PRINT#2,T$;S$;N$;S$;L$;S$;M$
6040 GOSUB 1010 :REM: --> HIBAVIZSGALAT
6050 PRINT
6060 PRINT TAB(10);"▧ *** ";U$;" *** ▀"
6070 TH=100: GOSUB 9010 :REM: --> VARAKOZTATAS
6999 RETURN
```

Mind a pozicionálás, mind az írás után hibavizsgálatot hajt végre, és csak akkor megy tovább, ha nem következett be hiba.

A sikeres műveletvégzésről leadja a művelet során beállított U$ üzenetet, amelynek elolvasására TH hosszúságú időt hagy.

Az adatbekérésre beszúrás vagy módosítás esetén kerül sor. Először a lakás LL$ telefonszámát kéri be:

```
7010 PRINT
7020 PRINT "▧?▀ LAKAS= ";L$;"  :   *";
7030 INPUT "████";LL$
7040 :   IF LEN(LL$)>7  THEN GOTO 7020
7050 :   IF LL$="*" THEN GOTO 7110
7060 :   IF LL$<>L$ THEN L$=LL$
```

Ehhez megjeleníti a rekordban tárolt L$ telefonszámot, illetve üres rekord esetén a pikk jelet. Ha csillagot adunk meg, akkor ezt az eredeti L$ értéket hagyja meg a rekordban. Ha nem, akkor kicseréli az LL$ értékre.

Ugyanígy jár el az M$ eredeti, illetve MM$ módosított munkahelyi telefonnal is:

```
7110 PRINT
7120 PRINT "▨?▀ MUNKA= ";M$;"  :   *";
7130 INPUT "████";MM$
7140 :  IF LEN(MM$)>12 THEN GOTO 7120
7150 :  IF MM$="*" THEN GOTO 7999
7160 :  IF MM$<>M$ THEN M$=MM$
7999 RETURN
```

Mindkét esetben egyszerű adatellenőrzést hajt végre, a megengedettnél hosszabb adatot nem fogadja el, hanem újra kéri.

A rekord beolvasása hasonló a korábbi programjainkban megismert olvasó szubrutinokhoz, csak itt a szubrutint mindig a kereső eljárásból hívjuk. Az RL, RH rekordcímről olvas, az RC bájtpozíciójáról.

RC = 1 esetén csak a T$ állapotjelzőt, egyébként a K$ nevet, az L$ lakástelefont és az M$ munkahelyi telefont olvassa be:

```
8010 PRINT#15,"P"CHR$(2)CHR$(RL)CHR$(RH)CHR$(RC)
8020 GOSUB 1010 :REM: --> HIBAVIZSGALAT
8030 IF RC=1 THEN GOTO 8210
8110 :  INPUT#2,K$,L$,M$
8120 :  GOSUB 1010 :REM: --> HIBAVIZSGALAT
8130 GOTO 8999
8210 :  INPUT#2,T$
8999 RETURN
```

Mind a pozicionálás, mind az olvasás után hibavizsgálatot hajt végre, és csak sikeres művelet esetén megy tovább.

A várakoztató rutin sem új, már láttunk ilyet. A TIME gépi órát kérdezi le először egy T1, majd ciklikusan egy T2 időpontban mindaddig, amíg a két időpont között eltelt idő el nem éri a megadott (TH hatvanad másodpercnyi várakozási) időtartamot:

```
9010 T1=TI
9020 T2=TI
9030 :  IF T2-T1<TH THEN GOTO 9020
9999 RETURN
```

Ha a programot begépeltük, mentsük ki lemezünkre TELEFONKONYV néven.

```
1 REM:   TELEFONKONYV
2 REM:   ------------
101 REM:
102 REM: ----------------------------------------MEGNYITAS
110 S$=CHR$(13)
120 PRINT "▢"
```

```
 130 FOR I=1 TO 20
 140 :   PRINT TAB(10);"TELEFONKONYV"
 150 NEXT I
 160 TH=100: GOSUB 9010 :REM: --> VARAKOZTATAS
 170 OPEN 15,8,15
 180 OPEN 2,8,2,"TELEFONTORZS"
 200 REM:
 201 REM:
 202 REM: -------------------------------KARBANTARTAS
 210 PRINT "□"
 220 :   PRINT:PRINT "▩*■ NEV=   *▮■■▮";
 230 :   INPUT N$
 240 :   :   IF LEN(N$)>25 THEN GOTO 210
 250 :   :   IF N$="*"      THEN GOTO 910
 260 :   GOSUB 2010 :REM: --> LEKEPEZES
 270 :   :   IF RL<1        THEN GOTO 210
 280 :   :   IF RL>126      THEN GOTO 210
 310 :   GOSUB 3010 :REM: --> KERESES
 320 :   IF     TALALT THEN GOSUB 5010 :REM:
 330 :   IF NOT TALALT THEN GOSUB 4010 :REM:
 399 GOTO 210                        --> MODOSITAS
 900 REM:                            --> UJ FELVITEL
 901 REM:
 902 REM: -------------------------------------ZARAS
 910 CLOSE 2
 920 CLOSE 15
 930 PRINT "□"
 940 FOR I=1 TO 20
 950 :   PRINT TAB(10);"VISZONTLATASRA!"
 960 NEXT I
 999 END
1000 REM:
1001 REM:
1002 REM: ==============================HIBAVIZSGALAT
1010 INPUT#15,H,H$,T,B
1020 IF H<20 THEN GOTO 1999
1030 :   PRINT
1040 :   PRINT " REKORD: ";RL
1050 :   PRINT
1060 :   PRINT " HIBA   :   ";"▩ ";H$;" ■"
1110 :   CLOSE 2
1120 :   CLOSE 15
1130 :   U$="LEALLITVA"
1140 :   PRINT
1150 :   PRINT TAB(10);"▩ *** ";U$;" *** ■"
```

```
1199 :  STOP
1999 RETURN
2000 REM:
2001 REM:
2002 REM: ================================LEKEPEZES
2010 B$=LEFT$(N$,1)
2020 RL=(ASC(B$)-65)*5+1
2030 RH=0
2999 RETURN
3000 REM:
3001 REM:
3002 REM: ==================================KERESES
3010 MEGTELT=0
3020 TALALT =0
3030 SZ     =0
3110 FOR I=1 TO 5
3120 :  RC=1: GOSUB 8010 :REM: --> OLVASAS
3130 :  IF T$="-" THEN GOTO 3210
3140 :  IF T$="π" THEN GOTO 3310
3150 :  IF T$="+" THEN GOTO 3410
3199 :  GOTO 3510
3210 : : IF SZ=0 THEN SZ=RL
3299 :  GOTO 3510
3310 : :  IF SZ=0 THEN SZ=RL
3399 GOTO 3910
3410 : :  RC=2: GOSUB 8010 :REM: --> OLVASAS
3420 :  IF N$<>K$ THEN GOTO 3510
3430 : :  SZ=RL: TALALT=(SZ<>0)
3499 GOTO 3910
3510 :  RL=RL+1
3599 NEXT I
3910 RL=SZ
3920 MEGTELT=(SZ=0)
3999 RETURN
4000 REM:
4001 REM:
4002 REM: ==============================:UJ FELVITEL
4010 PRINT:PRINT:PRINT
4020 PRINT "▩←■ NINCS NYILVANTARTVA!"
4030 PRINT
4040 PRINT "▩*■ FELVIHETO? =I/N=  *";
4050 INPUT "▮▮▮▮";V$
4060 :  IF V$= "N" THEN GOTO 4999
4070 :  IF V$<>"I" THEN GOTO 4040
4080 IF MEGTELT THEN GOTO 4410
```

```
4100 REM:
4101 REM:
4102 REM: --------------------------------BESZURAS
4110 :   T$="+"
4120 :   L$="♠"
4130 :   M$="♠"
4140 :   GOSUB 7010 :REM==> ADATBEKERES
4210 :   U$="FELVIVE"
4220 :   GOSUB 6010 :REM==> FELIRAS
4299 GOTO 4999
4400 REM:
4401 REM:
4402 REM: --------------------------------MEGTELT
4410 :   PRINT:PRINT:PRINT
4420 :   PRINT TAB(17);"▨   ■"
4430 :   PRINT "  A TELEFONKONYV ▨ ";B$;
4440 :   PRINT " ■ BETUS OLDALA"
4450 :   PRINT TAB(17);"▨   ■"
4460 :   PRINT
4470 :   U$="MEGTELT"
4480 :   PRINT TAB(10);"▨ *** ";U$;" *** ■"
4490 :   TH=200: GOSUB 9010 :REM: --> VARAKOZTATAS
4999 RETURN
5000 REM:
5001 REM:
5002 REM: ================================MODOSITAS
5010 PRINT
5020 PRINT "▨-■ LAKAS: ";L$
5030 PRINT
5040 PRINT "▨-■ MUNKA: ";M$
5110 PRINT:PRINT:PRINT
5120 PRINT "▨*■ VALTOZAS VAN? =I/N=  *";
5130 INPUT "▮▮▮▮";V$
5140 :   IF V$="N"  THEN GOTO 5999
5150 :   IF V$<>"I" THEN GOTO 5120
5210 PRINT
5220 PRINT "▨*■ TORLENDO?       =I/N=  *";
5230 INPUT "▮▮▮▮";V$
5240 :   IF V$="N"  THEN GOTO 5310
5250 :   IF V$<>"I" THEN GOTO 5220
5257 REM:
5258 REM:
5259 REM: --------------------------------TORLES
5260 :   T$="-"
5261 :   N$="O"
5262 :   L$="O"
5263 :   M$="O"
```

```
5270 :   U$="TOROLVE"
5280 :   GOSUB 6010 :REM: --> FELIRAS
5290 GOTO 5999
5300 REM:
5301 REM:
5302 REM: ---------------------------------VALTOZTATA
5310 :   GOSUB 7010 :REM: --> ADATBEKERES
5320 IF LL$="*" AND MM$="*" THEN GOTO 5999
5330 :   U$="MODOSITVA"
5340 :   GOSUB 6010 :REM: --> FELIRAS
5999 RETURN
6000 REM:
6001 REM:
6002 REM: ====================================FELIRA
6010 PRINT#15,"P"CHR$(2)CHR$(RL)CHR$(RH)CHR$(1)
6020 GOSUB 1010 :REM: --> HIBAVIZSGALAT
6030 PRINT#2,T$;S$;N$;S$;L$;S$;M$
6040 GOSUB 1010 :REM: --> HIBAVIZSGALAT
6050 PRINT
6060 PRINT TAB(10);"▣ *** ";U$;" *** ■"
6070 TH=100: GOSUB 9010 :REM: --> VARAKOZTATAS
6999 RETURN
7000 REM:
7001 REM:
7002 REM: ================================ADATBEKERE
7010 PRINT
7020 PRINT "▣?■ LAKAS= ";L$;" :   *";
7030 INPUT "■■■■";LL$
7040 :   IF LEN(LL$)>7  THEN GOTO 7020
7050 :   IF LL$="*" THEN GOTO 7110
7060 :   IF LL$<>L$ THEN L$=LL$
7110 PRINT
7120 PRINT "▣?■ MUNKA= ";M$;" :   *";
7130 INPUT "■■■■";MM$
7140 :   IF LEN(MM$)>12 THEN GOTO 7120
7150 :   IF MM$="*" THEN GOTO 7999
7160 :   IF MM$<>M$ THEN M$=MM$
7999 RETURN
8000 REM:
8001 REM:
8002 REM: ====================================OLVASAS
8010 PRINT#15,"P"CHR$(2)CHR$(RL)CHR$(RH)CHR$(RC)
8020 GOSUB 1010 :REM: --> HIBAVIZSGALAT
8030 IF RC=1 THEN GOTO 8210
8110 :   INPUT#2,K$,L$,M$
8120 :   GOSUB 1010 :REM: --> HIBAVIZSGALAT
```

```
8130 GOTO 8999
8210 :  INPUT#2,T$
8999 RETURN
9000 REM:
9001 REM:
9002 REM: ========================== VARAKOZTATAS
9010 T1=TI
9020 T2=TI
9030 :  IF T2-T1<TH THEN GOTO 9020
9999 RETURN
```

Mielőtt elindítanánk a programot, hozzuk létre a "TELEFONTORZS" állományt. Töltsük be a RELGENERALAS programot, és indítsuk el:

```
RELATIV ALLOMANY LETREHOZASA

ALLOMANY  NEVE =? TELEFONTORZS
REKORD  HOSSZA =? 49
REKORDOK SZAMA =? 131
VEGJEL         =? π

LETREHOZVA
```

Ügyeljünk arra, hogy az adatok hibátlanok legyenek, különben a karbantartás nem fog működni.

Megjegyezzük, hogy azért határoztuk meg 131 rekordban az állomány méretét, hogy a létrehozó program a végjelet ne az adataink területére írja.

Most már betölthetjük a TELEFONKONYV programot, és elindíthatjuk. Vigyünk fel néhány rekordot. Ezek, mivel az állomány üres, beszúrásként fognak megjelenni. Például:

```
NEV=? BACSKAI

NINCS NYILVANTARTVA!
FELVIHETO? =I/N=? I
NEV=? BACSKAI

NINCS NYILVANTARTVA!
FELVIHETO? =I/N=? I
LAKAS= ↑ :? 777-888
MUNKA= ↑ :? 999-000
[illegible]
```

Ha már néhány rekordunk nyilván van tartva, kipróbálhatunk egy lekérdezést. Például:

```
NEV=? CSONGRADI
LAKAS: 666-888
MUNKA: 777-999/555

VALTOZAS VAN? =I/N=? N
```

Illetve egy módosítást:

```
NEV=? BIHARI
LAKAS: -
MUNKA: 222-444

VALTOZAS VAN? =I/N=? I
TORLENDO?     =I/N=? N

NEV=? BIHARI
LAKAS: -
MUNKA: 222-444

VALTOZAS VAN? =I/N=? I
TORLENDO?     =I/N=? N
LAKAS= - :? 111-333
MUNKA= 222-444 :? *
```

[illegible]

Majd töltsük fel egy betű területét úgy, hogy az teljesen megteljen. Például:

```
NEV=? ALFOLDI VI

NINCS NYILVANTARTVA!
FELVIHETO? =I/N=? I


A TELEFONKONYV  H  BETUS OLDALA
```

[illegible]

Töröljünk innen egy rekordot. Például:

Ezután újra kíséreljük meg annak a rekordnak a felvitelét, amelynek beszúrása a túlcsordulási terület telítettsége miatt sikertelen volt:

```
NEV=? ALFOLDI III
LAKAS: 111-222
MUNKA: 333-444

VALTOZAS VAN? =I/N=? I
TORLENDO?      =I/N=? I

NEV=? ALFOLDI VI

NINCS NYILVANTARTVA!
FELVIHETO? =I/N=? I
LAKAS= ♠ :? 111-222
MUNKA= ♠ :? 333-444
```

A felvitel most már sikerülni fog.

A relatív állományt is megnézhetjük. Ehhez állítsuk le a programot NEV="*" válasszal, majd töltsük be a RELOLVASAS-t, és indítsuk el a "TELEFONTORZS" állományra, az adatokat mindig a 3. bájtról olvasva:

```
RELATIV ALLOMANY OLVASASA

ALLOMANY=? TELEFONTORZS

BAJTPOZICIO=? 3

   1 : + : ALFOLDI I
   2 : + : ALFOLDI II
   3 : + : ALFOLDI VI
   4 : + : ALFOLDI IV
   5 : + : ALFOLDI V
   6 : + : BARANYAI
   7 : + : BACSKAI
   8 : + : BIHARI
   9 : π :
  10 : π :
  11 : + : CSONGRADI
  12 : π :
  13 : π :

MEHET =I/N=? N
```

Használhatjuk erre a célra a bájtonként olvasó RELELLENORZES-t is, de az lassabb lesz. Igaz viszont, hogy a hibásan felírt adatokat is le tudja olvasni. Ezért ha a TELEFONKONYV program szemmel láthatólag hibátlanul lefutott, inkább az előbbi, ha pedig megszakadt vagy más hibajelenséget észleltünk, akkor az utóbbi kiíróprogramot használjuk.

FIGYELEM!
A példában fiktív neveket és telefonszámokat használtunk. Esetleges egybeesésük valamilyen létező névvel vagy telefonszámmal pusztán a véletlen műve. Abban a valószínűtlen esetben, ha ez mégis bekövetkezne, az olvasók szíves elnézését kérjük.

Megjegyezzük, hogy a TELEFONTORZS nem helyettesíti a zsebnoteszünket. Ha egy adott személy telefonszámát keressük, előbb be kell kapcsolnunk a COMMODORE-t, a lemezegységet, a tévét; meg kell keresnünk a lemezünket, be kell töltenünk róla a programot, el kell indítanunk; majd csak ezután kérdezhetjük le a szóban forgó személy rekordját. Ez néhány percet vesz igénybe, míg ugyanezt a műveletet a zsebnotesz segítségével egy nagyságrenddel rövidebb idő alatt elvégezhetjük.

Mire jó akkor ez a nyilvántartás? Nos, elsősorban „zsebnotesz" készítésére, ha van olyan programunk, amely az állományt ki tudja listázni.

Mivel a nyilvántartás könnyen karbantartható, gyorsan és gyakran állíthatunk elő naprakész listát róla, ami szinte tetszés szerinti példányszámban sokszorosítható. Ezt ugyan nem célszerű zsebben hordani, de például munkahelyi használatra nagyon megteszi.

Itt elsősorban nem is a házi telefonkönyvre gondolunk, hanem a hasonló szerkezetű egyéb nyilvántartásokra, mint amilyenek például a vevőkről, szállítókról, bérlőkről, ügyfelekről stb. készíthetők.

A gépi nyilvántartás igazi előnyét akkor látjuk, ha az állományból válogatni is tudunk. Ez az, amit a gép a kézi válogatásnál nagyságrendekkel rövidebb idő alatt és pontosabban végez el.

A leképezési eljárások hátránya ugyanaz, mint a zsebnoteszé: a nevek nem egyenletesen oszlanak el, így egyes lapok betelhetnek, másokon viszont még bőven van hely. A legegyszerűbben azzal védekezhetnénk ellene, ha olyan nagy noteszlapokat választanánk, hogy normális felhasználás esetén a leggyakoribb betűk lapjai se telhessenek be. Ez a megoldás azonban még a zsebnoteszeknél sem gazdaságos, nemhogy a COMMODORE-nál, ahol a lemez tárolókapacitása olyan kicsi (legfeljebb 167132 bájt), hogy igencsak takarékoskodnunk kell a hellyel.

A noteszgyártók e problémát úgy oldották meg, hogy a különböző betűk számára különböző méretű helyet tartanak fenn. (Egyes betűknek egy vagy több lap, más betűknek csak fél vagy negyed lap jut.)

A számítógépes feldolgozásban ez azt jelenti, hogy a lapok mérete nem egyforma, azaz az egyes betűknek különböző számú rekordhelyet tartunk fenn. Ilyenkor, ha például egy X(26) tömbben tároljuk az egyes betűkhöz tartozó rekordok számát, az N$ név elsődleges C címe a:

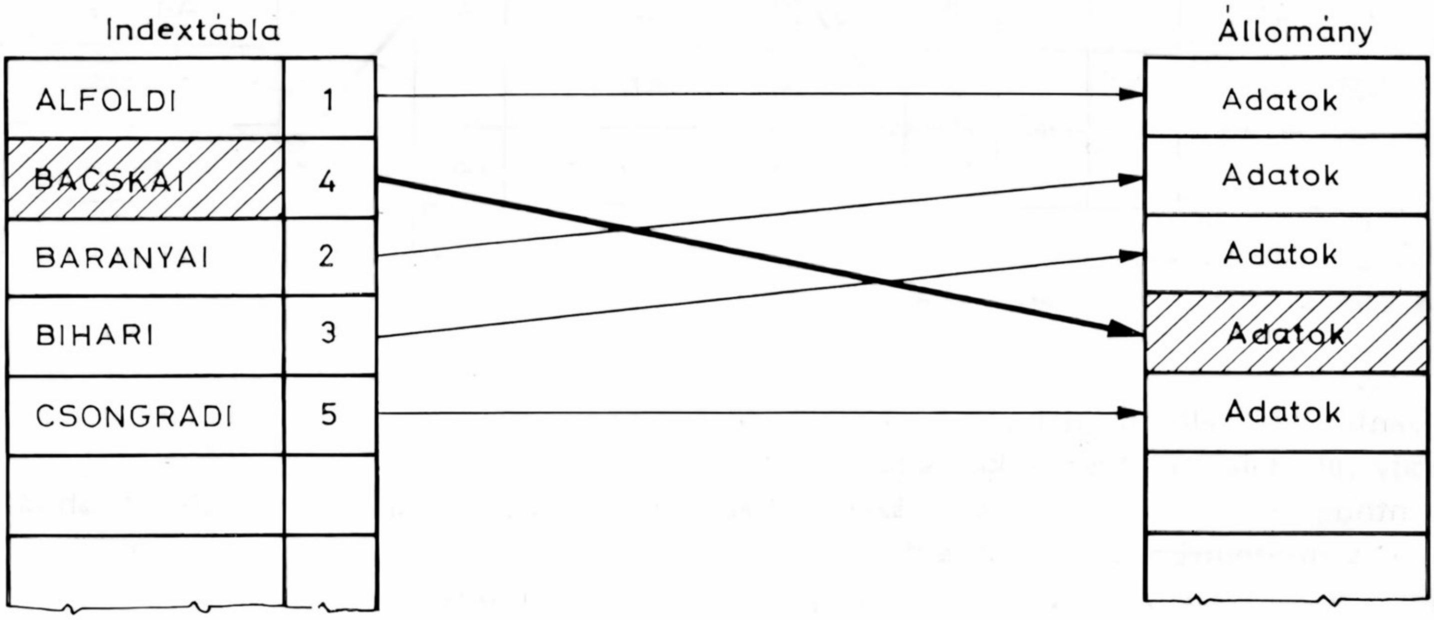
```
11010 C=1
11015 REM-----------------------------CIM KERESESE
11020 FOR I=65 TO 90
11030 :   IF ASC(N$)=I THEN 11999
11040 :   :   C=C+X(I-64)
```

```
11050 NEXT I
11055 REM---------------------------------HA NEM TALALTA
11060 C=0
11999 RETURN
```

leképező eljárással kapható meg. Alkalmasan megválasztott méretű lapok esetén a program a gyakorlatban is jól használható lesz.
Vannak azonban más lehetőségek is. Ezek közül kettőt megemlítünk az alábbiakban. Részletes tárgyalásuktól eltekintünk, mert az egyiket már részben ismerjük, a másik pedig jelentősen túlmutat a relatív állományok kezelésének alapismeretein.

RELATÍV ÁLLOMÁNY INDEXTÁBLÁVAL

Használhatjuk a random állományoknál már megismert indextáblát, amelyben a rekord azonosítóját és sorszámát tároljuk (43. ábra).



43. ábra. Relatív állomány indextáblával

Felvitelkor a rekordot érkezési sorrendben az állomány első szabad (törölt) helyére tehetjük, majd címét bejegyezzük a táblába.
A már tárolt rekordot úgy érhetjük el, hogy megkeressük az indextáblában az azonosítóját, és megkapjuk onnan a címét. Például TN$(N) nevekből és TC(N) címekből álló indextábla esetén egy N$ név C címe (rekordsorszáma) a:

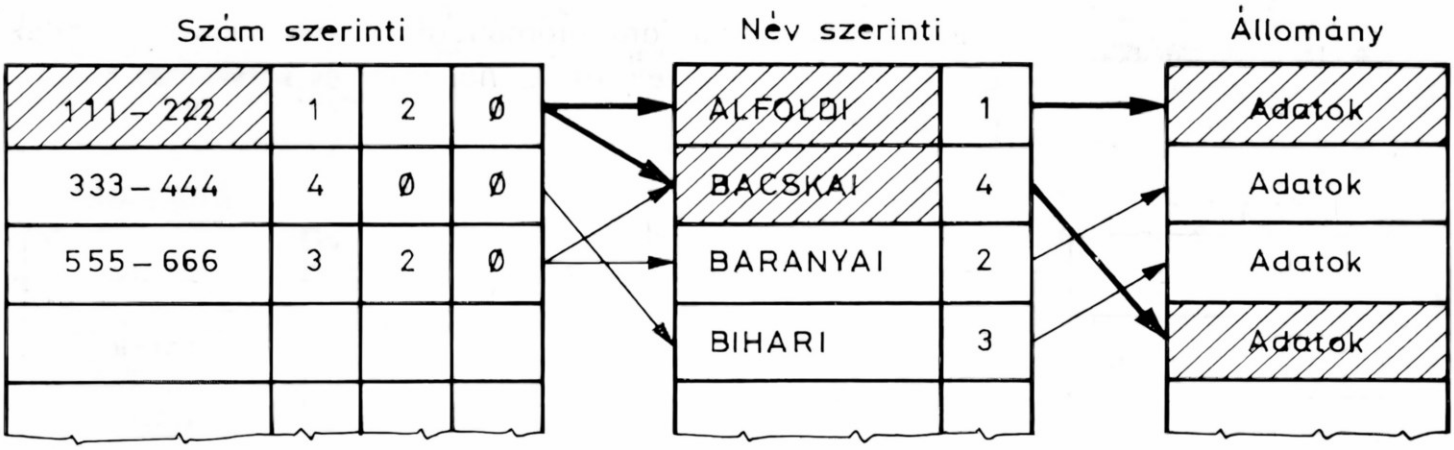
```
11005 REM-----------------------------------NEV KERESESE
11010 FOR I=1 TO N
11020 :   IF N$=TN$(I) THEN GOTO 11050
11030 NEXT I
11035 REM---------------------------------HA NEM TALALTA
11040 I=0
11045 REM----------------------------------HA MEGTALALTA
11050 C=TC(I)
11999 RETURN
```

eljárással határozható meg. A rekord a C címről beolvasható, illetve C=0, ha a rekord nem szerepel az állományban. (Kihasználtuk, hogy TC(0)=0.)

Ha az indextábla rendezetlen, kezelése egyszerűbb, de lassú benne a keresés, míg a rendezett indextáblában a gyors keresés ára a körülményesebb kezelés, hiszen a rendezettséget a beszúrásokkal nem ronthatjuk el.

A módszer előnye a nevek eloszlásától független, gazdaságos lemezkihasználás; az állományban nincsenek „holt" területek. Ezzel szemben viszont több az indextábla kezelésére fordítandó idő és tárhely.

Használhatunk két indextáblát is. Az egyikben név, a másikban telefonszám szerint tartjuk nyilván a rekordok címét (44. ábra).



44. ábra. Relatív állomány kettős indextáblával

Ilyenkor egy telefontudakozó szerepét betöltő program is írható, amely meg tudja adni, hogy valamely telefonszám kinek vagy kiknek a száma.

Minthogy egy telefonszám több személyhez is tartozhat, egy-egy bejegyzéshez több címet is tárolnunk kell. (Itt újra felmerül a kérdés, hogy mennyi helyet szánjunk erre.) Sőt még az is elképzelhető, hogy egy személynek több telefonszáma is van.

A telefonszám szerinti indextábla tartalmazhatja közvetlenül a rekordok címét is. Ez esetben a keresett személy nevét akkor tudhatjuk meg, ha az állományban tárolt adatok között szerepel. Ha ez az indextábla nem a rekordra mutat, hanem a név szerinti indextábla megfelelő bejegyzésére – mint az ábrán is látható, – akkor a kikeresett telefonszám bejegyzése alapján megkapjuk a nevet a név szerinti indextáblából, majd az onnan kivett rekordcím alapján, ha szükséges, hozzáférhetünk az állományban tárolt adatokhoz is. Például TT$(N) telefonszámokból TP(N,3) mutatókból, TN$(N) nevekből, TC(N) címekből álló indextáblák esetén az N$ névhez tartozó cím a már ismertetettt algoritmussal határozható meg, míg a T$ telefonszámhoz tartozó N$(3) nevek és C(3) címek az:

```
11005 REM-------------------------TELEFONSZAM KERESESE
11010 FOR I=1 TO N
11020 :   IF T$=TT$(I) THEN GOTO 11110
11030 NEXT I
11040 I=0
11105 REM-------------------------CIM ES NEV BETOLTESE
11110 FOR J=1 TO 3
11120 :   C (J)=TC (TP(I,J))
```

```
11130 :   N$(J)=TN$(TP(I,J))
11140 NEXT J
11999 RETURN
```

eljárással állíthatók elő. A tömbök feltöltetlen nulladik elemeit itt is kihasználva, C(J)=0 és N$(J)=" " értékeket kapunk az állományban nem szereplő rekordok, illetve PT(I,J)=0 esetén. Egyébként a rekordok a C(J) címekről olvashatók be, a nevek pedig az N$(J) tömbben találhatók.

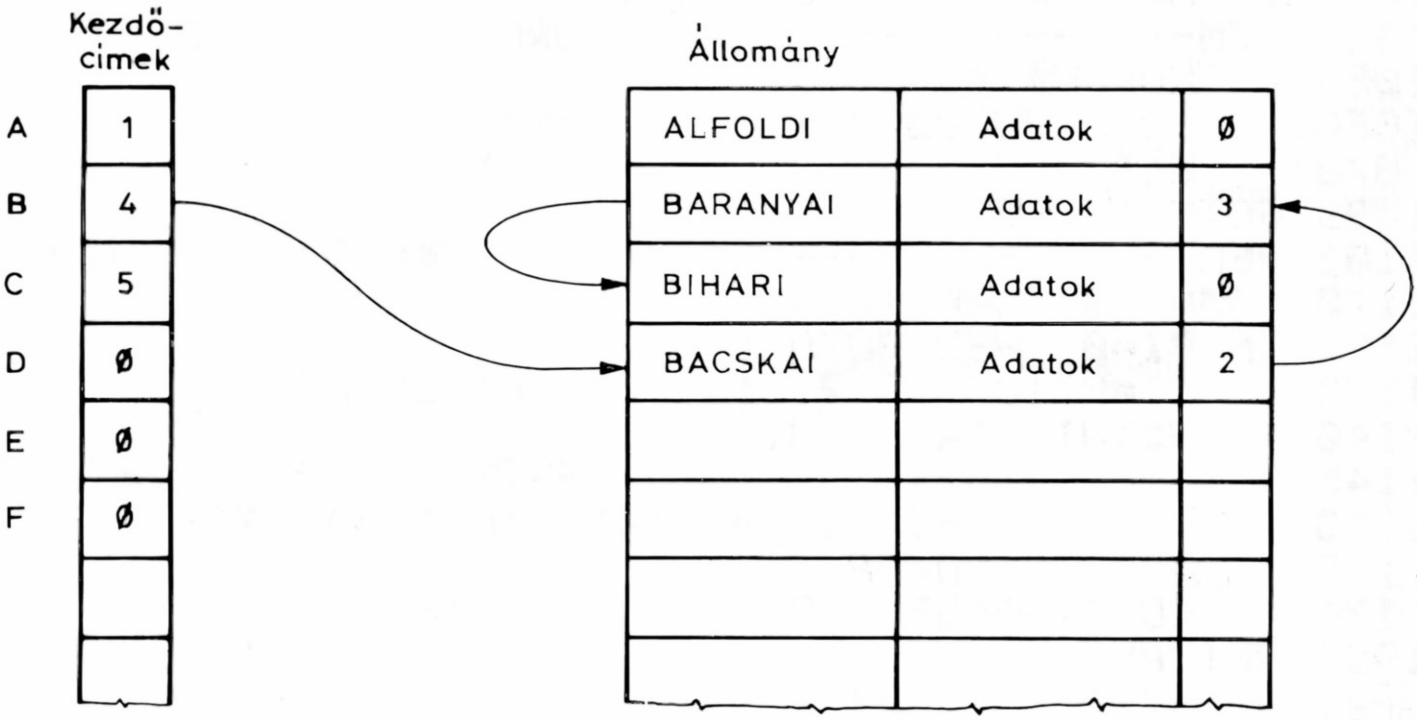
Ilyen szerkezetet akkor választunk, ha gyakran van szükség keresésre, viszont ehhez képest ritkán módosul az állomány. Vegyük észre, hogy bármilyen változás végrehajtása igen bonyolult, hiszen a két indextábla összhangját állandóan fenn kell tartanunk. Ehhez járul még az is, hogy a gyakori keresés rendezett indextáblák kezelését teszi indokolttá.

Ezzel még korántsem merültek ki a táblák lehetőségei. A bemutatottnál lényegesen hatékonyabbak is használhatók, mint ez a következő példában is látható.

RELATÍV ÁLLOMÁNY LISTASZERKEZETTEL

Már eddig is láthattuk, hogy a keresésben milyen szerepe van a rendezettségnek. A listaszerkezetek lehetővé teszik, hogy a rekordokat fizikai elhelyezkedésüktől függetlenül meghatározott logikai sorrendbe láncoljuk össze.

Ehhez nem kell más, mint hogy minden rekordban legyen egy mutató (pointer), amely az őt követő rekord címét tartalmazza (45. ábra):



45. ábra. Relatív állomány listaszerkezettel

Ilyenkor a rekordok tetszőleges, akár érkezési sorrendben lehetnek az állományban. Egy-egy betűn belül minden rekord tartalmazza az őt követő rekord címét, vagy ha ilyen nincs, akkor nullát. Célszerű, hogy külön táblázatban (indextáblában) tároljuk az egyes betűk elsődleges címét, hogy ne kelljen keresni őket.

E módszer előnye a tárolókapacitás gazdaságos kihasználása, amely független a nevek eloszlásától, és kezelésének egyszerűsége.

Ha például egy új rekordot kell beszúrnunk, mondjuk BEKESI néven, akkor ezt tárolhatjuk az első szabad helyen, mondjuk a 28-as címen. (Ehhez a címhez az állományban való kereséssel juthatunk, vagy tárolhatjuk az indextábla 0. elemében — ekkor nem kell keresnünk, de változás esetén gondoskodnunk kell a megfelelő módosításról.) Ezután végig kell menni a B betűsök listáján. Kiindulunk az indextáblából: az első B betűs a 4-es címen van. Ez BACSKAI. BEKESI névsor szerint ez után jön, tehát megnézzük a következő nevet, amely a 2-es címen található. BARANYAI is megelőzi BEKESI-t, így tovább megyünk a 3-as címre. Az ott talált BIHARI már BEKESI után jön, tehát megtaláltuk BEKESI logikai helyét. Most már nincs más dolgunk, mint hogy ennek megfelelően átállítsuk a mutatókat. BEKESI-nek BIHARI-ra kell mutatnia, tehát a rekordjában tárolt cím 3 lesz, míg BARANYAI-nak most már nem BIHARI-ra, hanem BEKESI-re kell mutatnia, így ebben a rekordban a 3-as címet át kell írni 28-ra. Ezután tárolható BIHARI rekordja a 28-as címen. (Végrehajtani könnyebb, mint elmondani.)

Legyen például E(26) az elsődleges címeket tartalmazó indextábla; NI$ és PI a CI címre beolvasott, NO$ és PO az állományba CO címre írandó név, illetve mutató (pointer). Ekkor az N$="BEKESI" beszúrása a C=28 címre a következő eljárással hajtható végre:

```
11010 I=ASC(N$)-64
11020 IF E(I)=0 THEN GOTO 11050
11025 REM-------------------ELSO LISTAELEM BETOLTESE
11030 :   CI=E(I):GOSUB 21010 :REM==> OLVASAS
11040 IF N$>NI$ THEN GOTO 11110
11045 REM-------------------BESZURAS A LISTA ELEJERE
11050 :   NO$=N$:PO=E(I)
11060 :   CO=C:GOSUB 31010 :REM==> IRAS
11070 :   E(I)=C
11099 GOTO 11999
11105 REM-----------------------KERESES A LISTABAN
11110 NO$=NI$:PO=PI:CO=CI
11120 IF PI=0 THEN GOTO 11150
11130 :   CI=PI:GOSUB 21010 :REM==> OLVASAS
11140 IF N$>NI$ THEN GOTO 11110
11145 REM-----------------------BESZURAS A LISTABA
11150 :   PP=PO:PO=C:GOSUB 31010 :REM==> IRÁS
11160 :   NO$=N$:PO=PP
11170 :   CO=C:GOSUB 31010 :REM==> IRAS
11999 RETURN
```

ahol a 21010-es szubrutin olvas egy rekordot a CI címről, a 31010-es pedig felírja a rekordot a CO címre.

Az sem gond, ha nem a lista közepébe kell beszúrnia egy új elemet, hanem a lista elejére vagy a végére. Erről a 11030—11070, illetve a 11120 sor gondoskodik. Sőt, a 11020 sor még az üres listába is lehetővé teszi az első elem felvitelét. (Az egyszerűség kedvéért az adatok kezelését nem vettük figyelembe.)

Hasonlóan egyszerű eljárások képezhetők a listaelemek módosítására, illetve törlésére is. A módszer hátránya, hogy egy betűn belül csak az első rekord érhető el az indextáblában tárolt elsődleges címen közvetlenül, az összes többi rekord csak a listaszerkezet vé-

gigjárásával, közvetlen elérések sorozatos végrehajtásával férhető hozzá. (Mellesleg már a túlcsordulási területet használó programoknak is megvan ez a hátrányuk.)
Előnyös viszont, hogy az állomány a lista mentén logikailag mindig rendezett. Megemlítjük még, hogy ez a listaszerkezet a lehető legegyszerűbb.
Képezhetők azonban úgynevezett cirkuláris listák is, amelyeknek utolsó listaeleme nem nulla címet (nullpointert) tartalmaz, hanem az első listaelemre mutat. Ilyenkor a lista többször is körbejárható. Van olyan lista, ahol a listaelemek nem a következő, hanem az előző elemre mutatnak. Ebben a listában hátulról előrefelé lehet haladni. Ha a listaelem két mutatót tartalmaz, vagyis mind az előző, mind a következő elem címét tároljuk benne, akkor kettős láncolású listát kapunk. Az ilyen listában tetszés szerint lehet előre vagy hátrafelé haladni. Az összetett listáknak olyan listaelemeik is lehetnek, amelyek maguk is listák.
Mindezek a tulajdonságok mind a relatív, mind a random állományoknál kihasználhatók. Részletes tárgyalásuk azonban nem ide tartozik. (A listákról és kezelésükről az érdeklődők bővebben olvashatnak dr. Mérey András már korábban idézett, ,,ADATSZERKEZETEK'' című könyvében.)

A RELATÍV ÁLLOMÁNY KEZELÉSÉNEK ÖSSZEFOGLALÁSA

A relatív szervezési mód alapja az, hogy ha egy állomány meghatározott számú és azonos méretű rekordokból áll, akkor a rekordok sorszámmal egyértelműen azonosíthatók, mivel mindig pontosan kiszámítható, hogy az állomány által elfoglalt területen hol helyezkedik el egy adott sorszámú rekord.
A relatív állomány csak előre meghatározott méretű és megfelelően előkészített lemezterületen hozható létre. A már létező állomány azonban tetszés szerint bővíthető, az alábbi korlátokkal: a relatív állomány legfeljebb 254 bájt méretű rekordokból állhat, amelyek együttesen nem foglalhatnak el 167132 bájtnál nagyobb területet a lemezen. A rekordok száma azonban semmiképpen sem haladhatja meg a 65535-öt.

A relatív állomány sajátosságai:

— A rekordjai folyamatosan, egymás után helyezkednek el a kijelölt lemezterületen, tekintet nélkül a szektorhatárokra, illetve sávhatárokra.
— A rekordjai a rekordsorszámmal azonosítva közvetlenül elérhetők.
— A relatív állományból rekordot törölni nem lehet, azaz az állomány nem szűkíthető.
— Kezeléséhez az állomány számára kijelölt adatcsatornán kívül szükségünk van a parancscsatornára is.

Az állománykezelő utasítások — az OPEN, POSITION, PRINT, INPUT, GET, CLOSE — másként működnek új állomány létrehozásakor, mint egy meglévő állomány használata során.

— Az állomány létrehozásakor:

OPEN: az állomány megnyitása létrehozásra

```
OPEN 15,8,15
OPEN állomány,8,csatorna,"állománynév,L"+CHR$(rekordméret)
```

Hatása: megnyitja a parancscsatornát, majd az állomány adatcsatornáját. A lemeztérkép alapján helyet keres az állománynak a lemezen. Bejegyzi a nevét a lemez tartalomjegyzékébe. Például egy 60 karakteres rekordokból álló állomány létesítése esetén:

```
OPEN 15,8,15
OPEN 2,8,2,"MINTA,L"+CHR$(60)
```

A rekordméret bájtokban értendő, legfeljebb 254 lehet, és ebbe beszámítanak az adatok közötti esetleges szeparátorjelek is.

POSITION: az állomány méretének meghatározása

```
PRINT#15,"P"CHR$(c)CHR$(a)CHR$(f)CHR$(1)
```

Ahol c=adatcsatorna, a=alsó bájt, f=felső bájt. Az utóbbi két paraméter együttesen az állomány méretét, az utolsó rekordjának sorszámát adja, az N=f*256+a képlet szerint.

Hatása:. kijelöli az N., az utolsó rekord helyét a lemezen. Például egy 1000 rekordból álló állomány esetén:

```
PRINT#15,"P"CHR$(2)CHR$(232)CHR$(3)CHR$(1)
```

A POSITION parancsot mindig a parancscsatornára kell kiírni egy külön PRINT utasítással

PRINT: a lemezterület előkészítése

```
PRINT#állomány,végjel
```

Hatása: a relatív állomány első N—1 rekordjának első bájtjába binárisan CHR$(255), azaz hexadecimális "FF" értéket tölt be, a rekord fennmaradó részébe pedig bináris nullák kerülnek. Az utolsó, N. rekordba viszont az első bájttól kezdődően a megadott végjel, majd egy CHR$(13) szeparátorjel kerül, és ezt követik a bináris nullák. Például:

```
PRINT#2,"VEGE"
```

Az állományazonosító a relatív állományé, a végjelet pedig tetszésünk szerint adhatjuk meg. A RETURN nem kerül a rekordba, ha a PRINT utasítást pontosvesszővel fejezzük be.

CLOSE: a létrehozott állomány lezárása

```
CLOSE állomány
CLOSE 15
```

Hatása: felszabadítja a relatív állomány számára lefoglalt adatcsatornát, majd a parancscsatornát. Például:

```
CLOSE 2
CLOSE 15
```

— A relatív állomány használatakor:

OPEN: a már létező állomány megnyitása

```
OPEN 15,8,15
OPEN állomány,8,csatorna,"állománynév"
```

Hatása: lefoglalja a parancscsatornát. Megkeresi az adott nevű állományt a lemez tartalomjegyzékében. Lefoglalja számára az adatcsatornát. Például:

```
OPEN 15,8,15
OPEN 2,8,2,"MINTA"
```

Ez a megnyitás csak már létrehozott relatív állományokra használható.

POSITION: pozicionálás adott rekord adott bájtjára

```
PRINT#15,"P"CHR$(c)CHR$(a)CHR$(f)CHR$(p)
```

Ahol c=a relatív állomány adatcsatornája, a=alsó bájt, f=felső bájt, p=bájtpozíció. Az alsó bájt és a felső bájt együttesen a rekordsorszámot adja meg, az N=f*256+a képlet szerint. A bájtpozíció pedig a beolvasandó vagy felírandó adat karakterpoziciója a rekordban, 1 és rekordméret—1 között.

Hatása: megkeresi a megadott rekord helyét a lemezen, és ha megtalálta, beáll a megadott bájtpozícióra. Például pozicionálás a 321. rekord 45. bájtjára:

```
PRINT#15,"P"CHR$(2)CHR$(65)CHR$(1)CHR$(45)
```

A relatív állomány bármely rekordjának kezelése, tehát minden írás és olvasás előtt kötelező pozicionálást végrehajtani. Ilyenkor a parancscsatornára kiadott PRINT utasításból egyetlen paraméter sem hagyható el, még akkor sem, ha az értéke nulla.

INPUT: a pozicionálás ellenőrzése

```
INPUT#15,hibakód,hibaüzenet,sáv,szektor
```

Hatása: ha a pozicionáló utasítás a rekordot megtalálta, a hibakódban 0, a hibaüzenetben pedig "OK" értéket kapunk. Ha a rekord azért nem volt megtalálható, mert nem szerepel az állományban, akkor a hibakódban **50,** a hibaüzenetben pedig **"RECORD NOT PRESENT"** érték jelenik meg. (A sávnak és a szektornak itt nincs lényeges szerepe.) Például:

```
INPUT#15,H,H$,SA,SZ
```

Az 1 és 19 közötti hibakódok nem jelentenek hibát, ugyanúgy tekintendők, mintha 0 hibakódot kaptunk volna. Minden más hibakódérték azonban sikertelen lemezműveletre utal. (A hibaállapot más lemezművelet után is lekérdezhető.)

PRINT: adatok írása a rekordba

```
PRINT#állomány,adatok
```

Hatása: ugyanúgy működik, mint a soros állományoknál, de az adatok felírása az előzőleg pozicionált rekordnak a pozicionálásban megadott bájtján kezdődik. Például:

```
PRINT#2,A$;S$;A%;S$;A
```

Ha olyan rekordra adunk ki PRINT utasítást, amely az állományban nem szerepel, azaz amelynek pozicionálására a hibakód 50 értéket vett fel, az állomány kiterjesztése automatikusan bekövetkezik. Ez azt jelenti, hogy az állomány utolsó rekordja és a megadott rekord közötti rekordok ugyanúgy feltöltődnek hexadecimális "FF" és "00" értékekkel, mint létrehozáskor. Ezután hajtódik végre a PRINT, amely ezután írja fel a megadott adatokat a rekordba.

INPUT/GET: adatok/bájtok olvasása a rekordból

```
INPUT#állomány,változók
GET#állomány,szöveges változók
```

Hatásuk: ugyanúgy működnek, mint a soros állományoknál, de az adatok, illetve bájtok beolvasása az előzőleg pozicionált rekordnak a pozicionálásban megadott bájtján kezdődik. Például:

```
INPUT#2,A$,A%,A
```

vagy

```
GET#2,E$
```

CLOSE: a használt állomány lezárása

```
CLOSE állomány
CLOSE 15
```

Hatása: felszabadítja az állomány számára lefoglalt adatcsatornát, majd a parancscsatornát is. Például:

```
CLOSE 2
CLOSE 15
```

A relatív állományok kezelése ennek megfelelően más és más stratégiát követ létrehozáskor és a meglévő állomány használatakor.

Létrehozás esetén:

— OPEN : Megnyitjuk az állományt létrehozásra, megadva az állomány nevét és a rekordméretet.
— POSITION : Pozicionálunk az állomány utolsó rekordjára, meghatározva ezzel az állomány méretét.
— PRINT : Felírunk egy tetszőleges adatot (célszerűen végjelet) az állomány utolsó rekordjába.
— CLOSE : Lezárjuk az állományt.

Megjegyeztük, hogy a relatív állomány méretét a létrehozáskor úgy célszerű meghatározni, hogy kiterjesztésére lehetőleg soha ne legyen szükség.

Állomány használata esetén:

- OPEN : Megnyitjuk az állományt, csak a nevét megadva.
- POSITION : A rekordsorszám és a bájtpozíció beállításával meghatározzuk, hogy a soron következő I/O művelet melyik rekord hányadik bájtpozícióján kezdődjék.
- INPUT : A parancscsatornáról lekérdezzük a pozicionálás sikerességét.
- INPUT : Megfelelő pozicionálás után bármely rekord bármely bájtpozíciójáról olvashatunk le adatokat.
- GET : Megfelelő pozicionálás után bármely rekord bármely bájtpozíciójáról olvashatunk le bájtokat.
- PRINT : Megfelelő pozicionálás után bármely rekord bármely bájtpozíciójára írhatunk fel adatokat, akkor is, ha ez az állomány területének (sikeres) kiterjesztésével jár.
- CLOSE : A feldolgozás befejeztével lezárjuk az állományt.

A relatív állomány a SCRATCH paranccsal törölhető, ugyanúgy mint bármelyik soros állomány vagy program.

Minthogy rekordot fizikailag törölni a relatív állományból nem lehet, gondoskodnunk kell arról, hogy az állomány „élő'' rekordjait meg tudjuk különböztetni a használaton kívüliektől. Ezért a törlendő rekordot vagy a létrehozáskor generált feltöltetlen rekorddal azonos tartalmú (úgynevezett üres) rekorddal írjuk felül, vagy a rekordban tárolt jelzővel (törlőjellel) határozzuk meg a rekord aktív vagy törölt állapotát. Az előbbi esetben a törölt adatok megsemmisülnek, az utóbbiban (a rekordban) megmaradnak.

A programozónak kell gondoskodnia annak számontartásáról, hogy melyik rekordnak mi a sorszáma, ha a rekordokban nem szerepel olyan adat, amely természetes rekordsorszámként közvetlenül használható, vagy amelyből a rekordsorszám egyértelműen képezhető. Szintén a programozó dolga a természetes vagy a leképezés során keletkező szinonimák kezelése.

Irodalom

Angerhausen–Becker–Gerits–Schellenberg: A BASIC programozás magasiskolája a C64-esen. DATA BECKER–NOVOTRADE, Budapest, 1985.

Bakó András dr.: Commodore 64 mikrogép programozása. Budapest, 1983.

Bodor Tibor: A Commodore 64 programozásának gyakorlata. SOROS LEMEZÁLLOMÁNYOK. SZÁMALK, Budapest, 1987.

Bodor Tibor: Commodore 64 felhasználói segédlet. INTRONIK, Budapest, 1984.

Bodor Tibor: Commodore 64 lemezkezelés (hallgatói segédlet). INTRONIK, Budapest, 1984.

Bodor Tibor–Gerő Péter: A BASIC programozás technikája. SZÁMALK, Budapest, 1983.

Bodor Tibor–Gerő Péter: A Commodore 64 programozásának gyakorlata. ALAPISMERETEK, SZÁMALK, Budapest, 1985.

Bodor Tibor–Gerő Péter: Bevezetés a korszerű FORTRAN programozásba. SZÁMALK, Budapest, 1983.

Bodor Tibor–Gerő Péter: MIKROSZÁMITÓGÉPEK (felhasználói segédkönyv az azonos című oktatófilm-, videókazetta- és diaképsorozathoz). NOVOTRADE–DELTASOFT, Budapest 1985.

COMMODORE GmbH: Commodore 64 Bedienungshandbuch. Frankfurt

COMMODORE GmbH: Commodore 64 MicroComputer Handbuch. Frankfurt

COMMODORE GmbH: Floppy Disk VC 1541 Bedienungshandbuch. Frankfurt

COMMODORE GmbH: VC 1541 Floppy Disk Bedienungshandbuch. Frankfurt

COMMODORE Inc: VIC–1541 Floppy Disk User's Manual. 1982.

COMMODORE Ltd.: Commodore 1541 Disk Drive User's Guide. Westchester, 1982.

Gerő Péter: Commodore 64 BASIC (hallgatói segédlet). INTRONIK, Budapest, 1984.

Lángos István: A Commodore 64 mikrogép kezelése és programozása. COMPORGAN, Budapest

Mérey András dr.: Adatszerkezetek. SZÁMOK, Budapest, 1979.

NOVOTRADE RT.: Commodore 64 felhasználói kézikönyv. Budapest

Ury László dr.: Commodore 64 BASIC felhasználói kézikönyv. LSI ATSZ, Budapest, 1984.

Ury László dr.: Commodore 64 Információs kártya. LSI ATSZ, Budapest

Commodore VIC–1541 floppy disk felhasználói kézikönyv.

Tárgymutató

A

B

L

M

N

O

P

O

R

Tibor Bodor

COMMODORE PROGRAMMING IN PRACTICE

RANDOM AND RELATIVE FILES

The Commodore–64 has two different facilities for the direct access of data stored on disk. This volume has been written to present these file organization modes; it consists of two parts, discussing the random and the relative files respectively.
The description of the file handling statements occupy altogether only two subchapters, one for the random, another for the relative files. The remaining parts of the book present different methods for the usage of these files; the file, record and data handling techniques, and the effects of direct access to the electronic data processing.
The different data processing functions, such as generating, creating, updating, processing, testing of files, and the programming technologies for handling them are illustrated by several examples and executable programs, including detailed discussions of random files using index tables and relative files containing overflow area.

Kiadja: a Számítástechnika-alkalmazási Vállalat
Felelős kiadó: Havass Miklós vezérigazgató
Felelős szerkesztő: Dr. Brückner Huba
Műszaki vezető: Molnár Zoltán
Műszaki szerkesztő: G. Müller Zsuzsa
A fedelet tervezte: Molnár Zoltán
Az ábrákat rajzolta: Olgyai Gézáné
Megjelent: 16,75 (A/5) ív terjedelemben

87/1238 – Vízügyi Dokumentációs Szolgáltató Leányvállalat nyomdája
Bp., VII., Kazinczy u. 3/b
Felelős vezető: Kaj István

Ára: 137,-Ft